Flash Action Script 3.0 Handbook

宮田亮 Miyata Ryo

Flash Action Script 3.0 宮田亮 Miyata Ryo Handbook

SoftBank Creative

SoftBank Creative

Flash Action Script 3.0 Handbook

宮田亮 Miyata Ryo

■サンプルのダウンロード

本書内で使用しているサンプルは以下のURLからダウンロードできます。

http://isbn.sbcr.jp/59268/

はじめに

デザイナーツールとしてのFlash、RIAインフラとしてのFlexというカテゴリがよく説明されていた状況はAdobe Flash Platformの登場により、そのルーツを同じくするものとして統一されました。AdobeがプロモートするFlash Platformはツールやプラットフォーム、開発インフラという提供する側だけでなく、ビジネスパートナーやユーザコミュニティを含むエコシステムという壮大な構想を持っています。目指すものはWeb、デスクトップ、モバイルをはじめとするあらゆるメディアへのFlashテクノロジーの浸透です。

本書で解説するActionScript3.0 Flash APIは、Flash Platformの心臓部であるFlash Playerが提供するAPIに直接アクセスし、もっとも効率よく稼動させる手段です。本書では、その主な機能をジャンル別に分類し、サンプルコードと共に解説します。

Flash Player10で実現される強力な機能としては、3D操作、高度なテキストレイアウトなどが筆頭に挙げられます。3D操作に関しては、難解な座標に関する数学処理をいっさい理解する必要なしに高度な3D処理を追加できます。高度なテキストレイアウトとは、25年の歴史を持つAdobeのタイポグラフィテクノロジーがFlash Playerに融合されたもので、Flash Platformがサポートするメディアに印刷品質と同等のパブリッシングを提供するものです。流し込まれたテキスト(フロー)に対する複数コラム、双方向文字、縦書き文字、その他の高度な制御を提供します。

Flash Platformの統一によりもたらされるFlashコーダーへのもう1つのアドバンテージは、Flash Builderというプログラム開発統合環境との融合が行われたことです。Flash BuilderはFlex開発ツールであるFlex Builderの最新バージョンですが、本書で解説するように、Flash CS5プロジェクトを作成編集、デバッグ、ActionScriptプロジェクトの作成とデバッグが可能となっており、ActionScriptを駆使するFlashコーダーの開発効率を向上させることが強く期待できます。

ページ数などの都合により、最新のFlash APIが持つすべての技術要素を解説することはできませんが、もっともベースとなる概念と使用すべきクラスとそのメソッド、プロパティについて、できるだけ多く掲載しました。全体を通してお読みいただくか、開発の傍(かたわ)らに置かれ、Flash APIの本質を理解され、Adobeが提供するプラットフォームを使用して、どれだけ高度なパフォーマンスを実現できるかもご理解いただければ幸いです。

2010年6月

宮田 亮



CONTENTS

PART 1
ActionScriptプログラム作成の基本

PART 5
テキストと正規表現処理

PART 6
高度なテキスト処理

PART 7
ビジュアルエレメントの操作

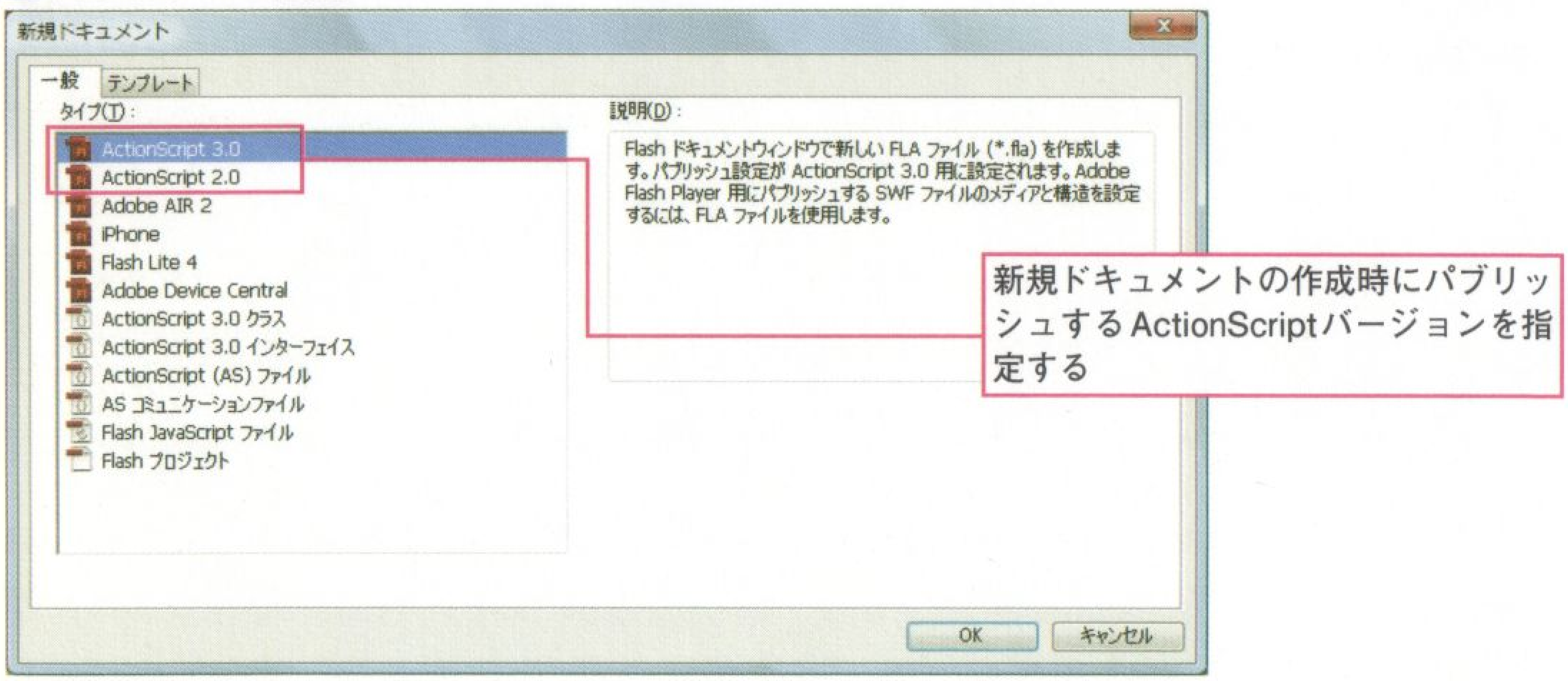
PART 1

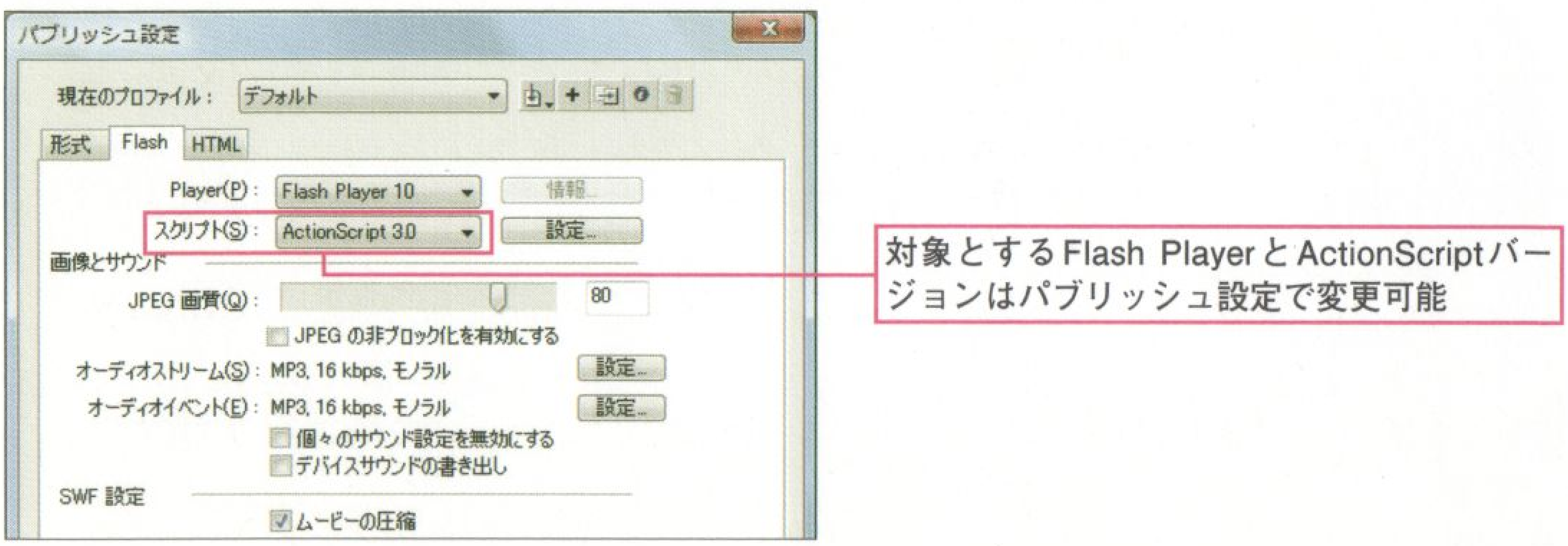
ActionScriptプログラム作成の基本

1-1-1 ActionScriptバージョンとFlash Player、AVMの互換性を確認する

Flashのスクリプト言語として進化してきたActionScriptですが、Version3.0では本格的なオブジェクト指向に基づくコーディングができるようになりました。また実行環境であるFlash Player内のAVM（ActionScript Virtual Machine）もバージョンが上がり（AVM2）、パフォーマンスが大幅に改善されています。ActionScript3.0でプログラミングを行う人は、旧バージョンのActionScriptおよびAVMの組み合わせによりどのような機能がサポートされるかを把握しておく必要があります。新規にFlashアプリケーションを作成する際には、パブリッシュをActionScript3.0で行うか、ActionScript2.0で行うかを選択します。ActionScript2.0を選択した場合、ActionScript3.0のAPIを使用するコードを記述するとコンパイルエラーになります。ただしこの設定は固定的なものではなく、パブリッシュ設定のスクリプトをActionScript3.0に変更することで解決できます。

■ パブリッシュ設定





Flash Playerに含まれるAVMバージョンと実行可能ActionScriptバージョン

Flash Player	AVMバージョン	ActionScriptバージョン
Flash Player 7	AVM1	1.0, 2.0
Flash Player 8	AVM1	1.0、2.0
Flash Player 9	AVM1、AVM2	1.0、2.0、3.0
Flash Player 10	AVM1、AVM2	1.0、2.0、3.0

ActionScript3.0とFlash Player10に関する互換情報

トピック	内容
Flash Playerの旧バージョン用に書かれたアプリケーションの実行	Flash Playerはバージョンアップの際に常に旧バージョンとの互換性を維持します。そのため、Flash Player10より以前に作成されて実行されているSWFはFlash Player10でロード可能です。
ActionScriptバージョンの混在	ActionScript2.0で作成したアプリケーションにActionScript3.0コードを追加することはできません。つまり、1つのアプリケーション(SWF)内でActionScript1.0/2.0とActionScript3.0コードは混在できません。
SWFファイルのロード	ActionScript3.0で書かれたアプリケーションはActionScript1.0/2.0で書かれたSWFをロードできますが、内部の関数/変数にはアクセスできません。ActionScript1.0/2.0で書かれたアプリケーションはActionScript3.0のSWFをロードできません。

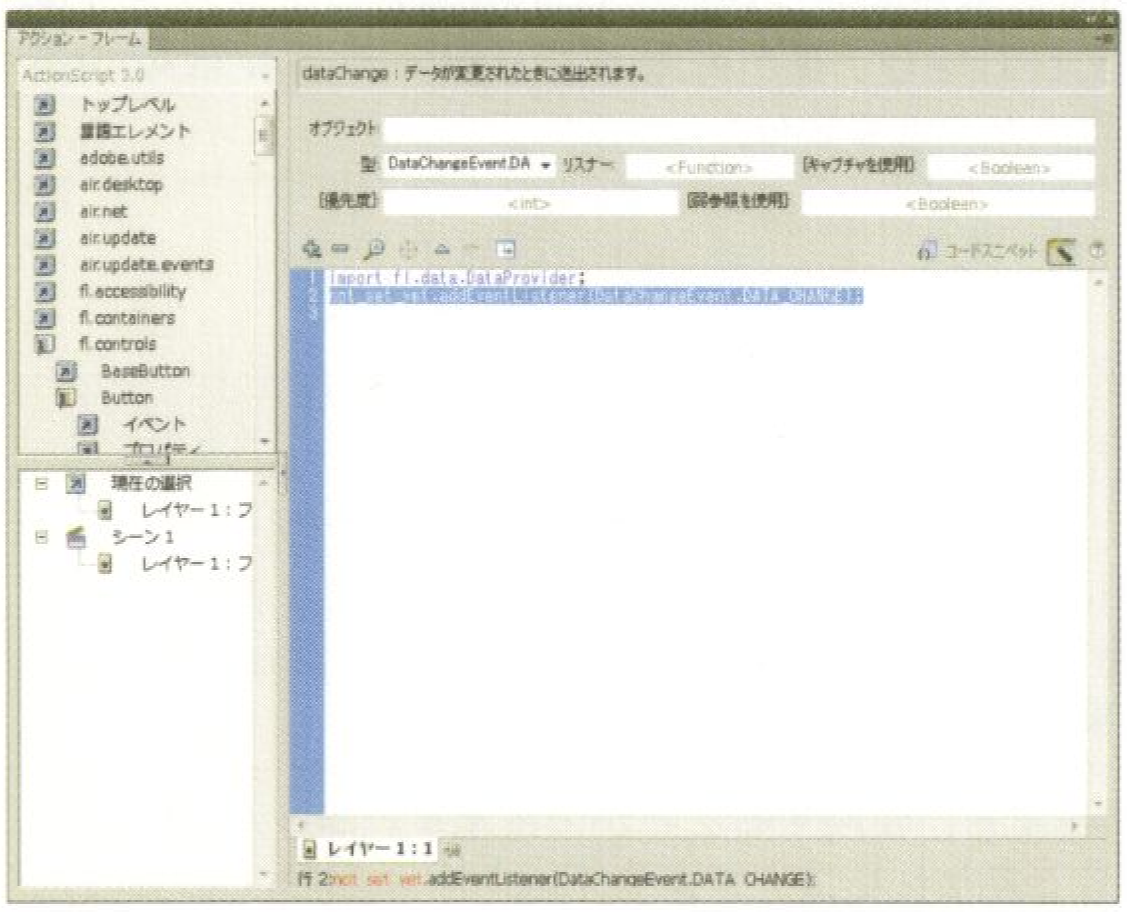
ActionScriptとFlash Liteのバージョン

バージョン	概要
ActionScript3.0	AVM2上で、以前のバージョンに比べて高速に実行されます。オブジェクト指向プログラミングの理解が必要になります。ECMAScript仕様に完全に準拠しています。ActionScript3.0を使用するFLAファイルには、以前のバージョンの ActionScriptを含めることができません。
ActionScript2.0	ActionScript3.0より簡単に学習でき、使用することができます。コンパイルされたコードは、ActionScript 3.0で書かれたコードより Flash Player上での実行速度は遅くなりますが、コンピュータに負荷がかからない種類のプロジェクトではActionScript2.0のほうが有利になる場合があります。ActionScript 2.0はECMAscript仕様に基づいていますが、完全に準拠しているわけではありません。
ActionScript1.0	ActionScriptのもっとも簡単な形式であり、Flash Liteプレーヤーの一部のバージョンで引き続き使用されています。ActionScript1.0および2.0は、同じFLAファイルに共存できます。
Flash Lite 2.x	携帯電話およびモバイルデバイスで実行されているFlash Lite 2.xによりサポートされるActionScript 2.0のサブセットです。
Flash Lite 1.x	携帯電話およびモバイルデバイスで実行されているFlash Lite 1.xによりサポートされるActionScript1.0のサブセットです。

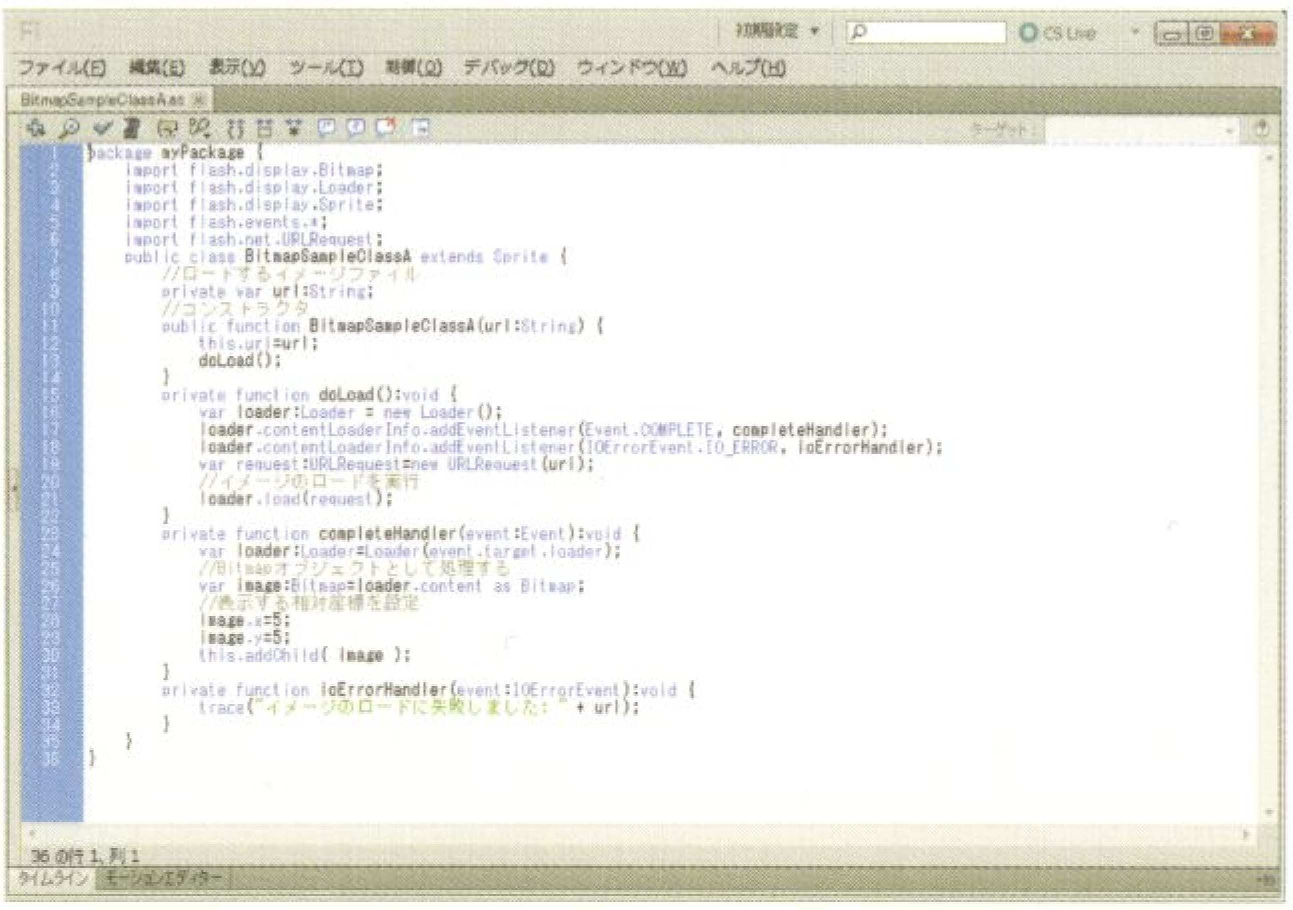
1-1-2 ActionScriptコードの作成、編集と保存

ActionScriptコードを作成、編集するためのアドビ製品には、Flash ProfessionalなどのFlashオーサリングツールとFlash Builderがあります。Flashオーサリングツールを使用する場合、フレームアクションはアクションパネル、外部カスタムクラスファイルやドキュメントクラスファイルの編集はスクリプトウィンドウで行います。アクションパネル［ウィンドウ］-［アクション］またはF9で開き、スクリプトウィンドウは、外部asファイルを開くと開始します。Flash Builderでは、ActionScriptプロジェクトとFlash CS5プロジェクトを作成して編集することができます。ActionScriptプロジェクトはFlash APIを使用してSWFを構築するためのプロジェクトですが、Flashオーサリングツールのようにステージのビジュアル編集はできません。Flashプロジェクトでは、Flash CS5で作成されたFLAファイルやXFLファイルを開いてデバッグしたり、Flash BuilderからFlash Professionalを起動してパブリッシュしたり、Flash Professionalのデバッガを起動することができます。

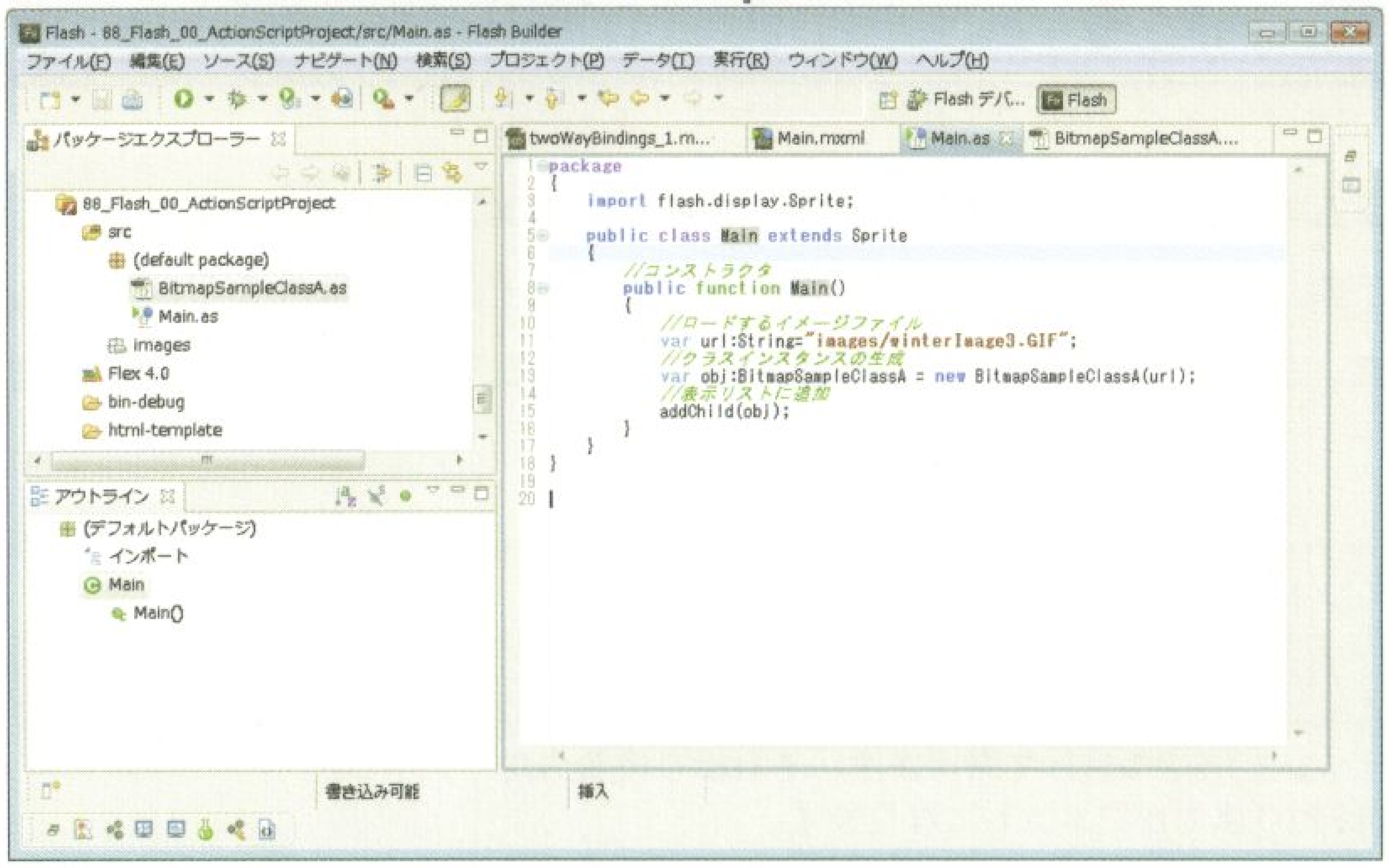
■ アクションパネル［ウィンドウ］-［アクション］またはF9で表示



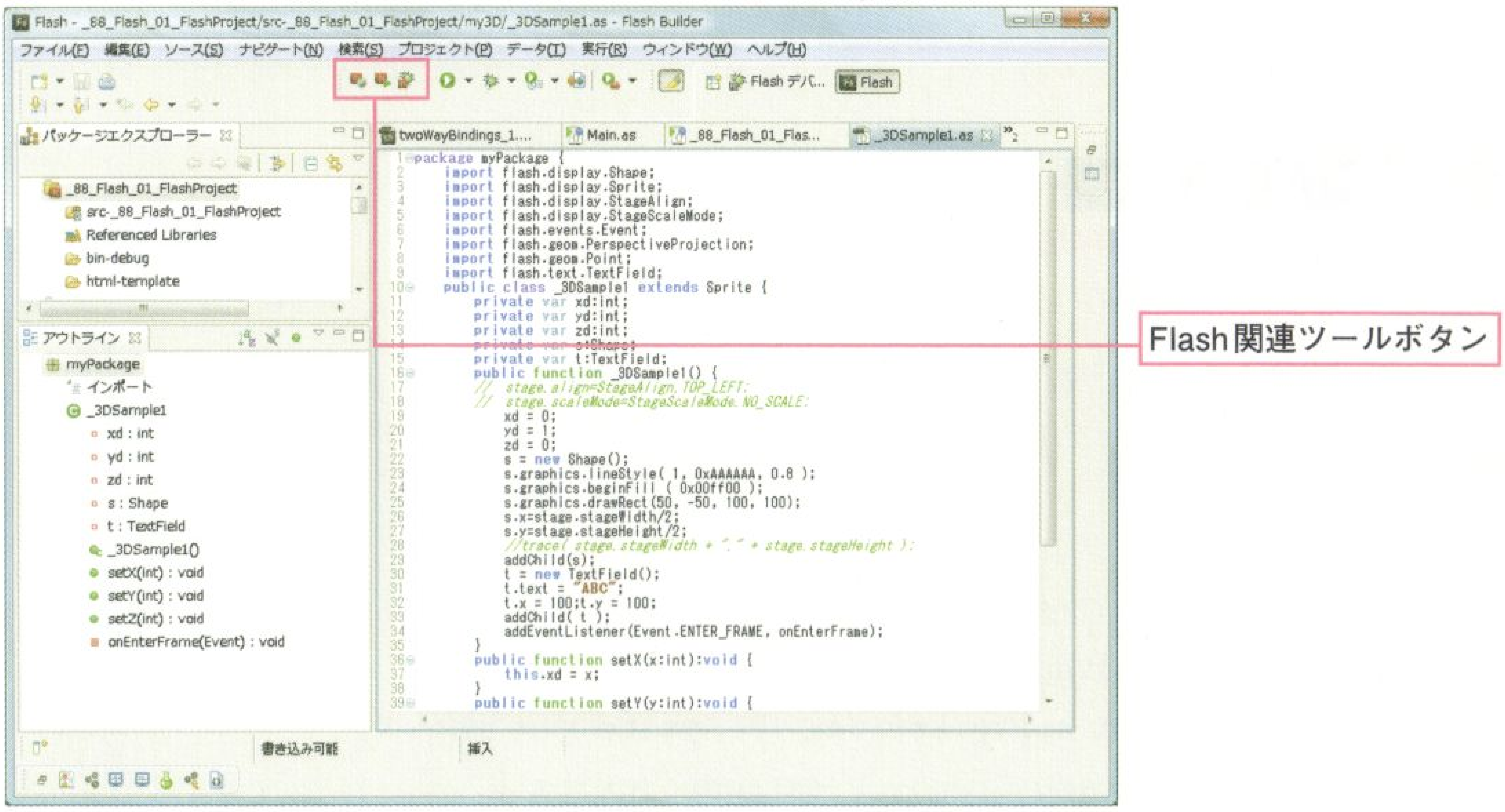
■ スクリプトウィンドウ



■ Flash BuilderによるActionScriptプロジェクトの編集



■ Flash BuilderによるFlash CS5プロジェクトの編集



■ Flashプロジェクトツールバーのボタン

ボタン	働き
	Flash Professionalでパブリッシュ
	Flash Professionalでムービーをプレビュー
	Flash Professionalでムービーをデバッグ

1-1-3 ActionScriptプログラムにコメントを書く

PART 1 2 3 4 5 6 7 8 9

ActionScriptプログラム中にコメントを書くことができます。コメントはプログラムとして処理されないテキストです。具体的には変数の用途や処理の概要などのメモ書きをプログラム中に書いておくものです。コメントを残しておくと他人がプログラムを理解するときにも役立ちます。

ActionScriptのコメント

種類	概要	サンプル
1行コメント	/(スラッシュ)を2つ続けて書きます。それより後ろの文は行の終わりまでがコメントになります。	// コメントテキスト
複数行コメント	/(スラッシュ)と*(アスタリスク)を組み合わせて書きます。開始部分は「/*」、終了部分は「*/」で書きます。開始部分と終了部分の間がコメントになり、複数行のコメントを書くことができます。	/* コメントテキスト ... */

SAMPLE CODE

ActionScriptプログラム中のコメント文　　BasicSampleA.fla

```
/*
    クラスの概要:このクラスはグラフィック処理を行うサンプルです。        ①複数行コメント
    コンストラクタ:引数をとりません
*/
package {
  // 必要なパッケージをインポートする                               ②1行コメント
    import flash.display.Shape;
    import flash.display.Sprite; // Spriteクラスを使用します        ③1行コメント
    public class test_Graphics extends Sprite {
      public function test_Graphics()         {
        var rect:Shape = new Shape();
        rect.graphics.beginFill( 0xFFFFCC );
        this.addChild( rect );
      }
  }
}
```

①が /* */ による複数行コメントです。この例では、クラスの概要を記述しています。②と③に1行コメントがあります。③のように、1行コメントはプログラム文の後ろに記述してその文の説明書きとする場合が多いです。

1-1-4 ActionScriptの値表現（リテラル）を理解する

プログラム中の値の書き方は、リテラル（Literal）と呼ばれます。例えば半角で「1」と書くと、ActionScriptコンパイラーは整数の1と解釈します。これは整数リテラルと呼ばれます。「"1"」と書くとストリング（String）の"1"と解釈されます。また、ActionScriptでは、XMLリテラルにより、プログラム中に直接XMLデータをコーディングできます。よく使用するリテラルをご紹介します。

ストリングリテラル

ActionScriptでは、"（二重引用符）か '（一重引用符）でテキストを囲むことでストリング値を表記できます。ストリングは一連の文字、数値その他の文字をまとめて1つの値として扱うためのものです。

```
trace("I'm a loser" );
trace('This is A String');
```

1行目では、"（二重引用符）、2行目では '（一重引用符）で囲むストリング値を作成してtrace出力しています。

▶ 実行結果

```
I'm a loser
This is A String
```

数値リテラル

ActionScriptでは、プログラム中に書かれた数値は整数（int または uint）またはNumber型の値として解釈されます。小数つきの数値はNumber型になります。

```
trace( 100 is int );
trace( 3.14 is Number );
```

1行目では100というint値を、2行目では3.14というNumber値を書いています。

▶ 実行結果

```
true
true
```

XMLリテラル

ActionScriptでは、XMLをストリングとして書く必要はなく、直接プログラム中に値として書くことができます。


```
var myXML:XML =
<product>
<item id='001'>
  <menuName>toro</menuName>
  <price>19.99</price>
</item>
</product>
trace( myXML.toString() );
```

2行目から7行目で書かれたXMLは1行目で宣言された変数myXMLに代入されます。

▶ 実行結果

```
<product>
  <item id="001">
    <menuName>toro</menuName>
    <price>19.99</price>
  </item>
</product>
```

配列リテラル

角かっこ「 [ ] 」の中に「,」(カンマ)を区切り文字として使用して配列を書くことができます。

```
var a1:Array = [ 'a', 'b', 'z' ];
Trace ( a1 );
```

1行目で3つの要素からなる配列を作成して、宣言された変数a1に代入します。

▶ 実行結果

```
a,b,z
```

オブジェクトリテラル

中かっこ「 { } 」の中にキーと値のペアを「,」(カンマ)を区切り文字として使用してObjectイ

ンスタンスを書くことができます。このようなインスタンスは結合配列とも呼ばれます。

```
var person:Object = { name: "Cameron", gender: "machine" };
trace( person.name + " is " + person.gender );
```


name、genderという2つのプロパティを持つObjectインスタンスを作成し、宣言されたperson変数に代入します。

▶ 実行結果

```
Cameron is machine
```

1-2-1 変数の宣言とデータ型

ActionScript3.0では、変数は必ずvarステートメントを使用して定義します。変数に格納するデータの種類はデータ型で表します。ActionScriptでは、データ型の宣言は必須ではありませんが、使用が推奨されます。データ型は、変数名の後ろに：（コロン）をつけて指定します。例えばtrue / false値を扱うための変数はBoolean（ブーリアン）型、日付データを処理するときはDate（デート）型の変数を宣言します。データ型は、格納する値によりプリミティブ型と複合型に分けられます。ActionScriptの変数名に使用できる文字は下記のように決まっています。また、変数名の大文字と小文字は別のものとして区別されます。例えば、myNameとMyNameは別の変数となります。

ActionScriptの変数宣言は、varの後に変数名（variableName）、続けて：（コロン）の後にデータ型（dataType）を書きます。

```
var variableName:dataType
```

ActionScriptの変数名として使用できる文字（下記の文字を組み合わせて変数名を定義できます）

- **a-z の英文字**
- **A-Z の英文字**
- **0-9 の数字（数字は変数名の先頭にはできません）**
- **$ _**

プリミティブデータ型

プリミティブデータ型はBoolean、int、Number、String、uintなどの「プリミティブ値」と呼ばれる基本的なデータ型です。プリミティブデータ型の変数には、単一のプリミティブ値を格納できます。

```
var i:int;
i = 0;
trace( i );
```

int型の変数 i を宣言して0というプリミティブ値を代入します。

▶ 実行結果

```
0
```


複合データ型

複合型はプリミティブ値以外の値を格納するデータ型で、複数の値をセットとして格納します。Array、Date、Error、Function、RegExp、XML、XMLList、ObjectなどのActionScriptコアクラスは代表的な複合データ型になります。

```
var o:Object = { name: "Arnold", gender: "machine" }
trace( o );
```

Object型の変数oを宣言して、nameとgenderというプロパティを設定します。この操作により、oは2つのプロパティを持つ「複合データ」になります。複合データの場合、trace()で直接出力すると下記のような結果になります。

▶ 実行結果

```
[object Object]
```

1-2-2 変数のデフォルト値を使う

PART 1

ActionScriptではデータ型の種類によって変数のデフォルト値が決まっています。デフォルト値は、変数を宣言すると自動的に設定される値です。

■ ActionScriptのデータ型とデフォルト値

データ型（日本語読み）	概要	デフォルト値
Boolean（ブーリアン）	true / falseの2値を格納できます。Boolean型の変数は、if 文の判定にも使用できます。	false
int（イント）	整数値を格納できます。ActionScriptでは、32ビットを使用するため、-2^{31} ～ $2^{31}-1$の範囲が有効です。この範囲を超える数値を扱う場合、Number型を使用します。	0
Number（ナンバー）	整数と小数つきの数値を格納できます。64ビットが使用されます。値の範囲は、Number.MAX_VALUE ～ Number.MIN_VALUEで調べることができます。	NaN
String（ストリング）	文字列を格納するデータ型です。UTF-16形式のUnicode文字コードで格納されます。	null
uint（ユーイント）	正（プラス）の整数値を格納できます。int型と異なり、マイナス値を扱えませんが、範囲が約2倍になります。0 ～ $2^{31}-1$の範囲に使用できます。uintのuは、unsigned（符号なし）を意味します。	0
Object（オブジェクト）	ActionScriptクラス階層のルートクラスです。つまり、すべてのクラスの基本クラスになります。	null
ActionScriptクラス ユーザ定義クラス	ActionScript APIで提供されるクラスやユーザ定義クラスを使用する場合、変数を宣言してインスタンスを生成します。	null
*（デフォルト）	変数の型指定がない場合、デフォルトのデータ型を表す * になります。	undefined

※ **NaN**は「Not A Number」で、IEEE-754で規定されている数値ではないことを表す特別な値です。

※ **undefined**は「初期化されていない、型指定のない変数」や初期化されていないダイナミックオブジェクトのプロパティに設定される特殊な値です。

※ 上記以外にnull値だけで構成されるNullデータ型やundefined値だけで構成されるvoidデータ型があります。nullは「値を設定していない」ということを表す特殊なデータ型で、関数の戻り値として「返すデータがない」ことを示すためにも使用されます。voidは関数の宣言で、値を返さないことを示します。

SAMPLE CODE

BasicSampleB.fla

```
var b:Boolean;      ─┐
var i:int;           │
var n:Number;        │
var s:String;        ├── ①変数の宣言
var ui:uint;         │
var o:Object;        │
var d:Date;          │
var undef;          ─┘
trace( "b=" + b );  ─┐
trace( "i=" + i );   │
trace( "n=" + n );   │
trace( "s=" + s );   ├── ②変数の値をtrace()出力
trace( "ui=" + ui ); │
trace( "o=" + o );   │
trace( "d=" + d );   │
trace( "undef=" + undef ); ─┘
```

①が変数の宣言です。②で、宣言した変数の値をtrace()出力します。それぞれの変数には値を代入していないため、デフォルト値が出力されます。

▶ 実行結果

```
b=false
i=0
n=NaN
s=null
ui=0
o=null
d=null
undef=undefined
```

1-2-3 定数を宣言して使う

PART
1
2
3
4
5
6
7
8
9

例えば円周率をよく使用するアプリケーションの場合、計算する場所で3.14という数値リテラルを使用するよりも、3.14という値を持つ変数を作成しておいたほうが効率の良いコーディングが行えます。この場合、その変数は固定値を持つ特殊な変数として作成します。このような変数は定数と呼ばれます。

ActionScriptの定数宣言は、constの後に定数名（CONSTANT_NAME）、続けて：（コロン）の後にデータ型（dataType）を書き、値を代入します。値はプログラム中で変更することはできません。定数は慣習的に大文字アルファベットと数値を使用し、単語の区切りを_（アンダースコア）でつないで構成しますが、これは定数名を見てその内容がわかるため効率的です。この命名方法は必須ではありません。また、ActionScriptには、システムで定義されている定数があり、プログラムで自由に参照できます。この定数はグローバル定数と呼ばれます。

ActionScriptの定数宣言

```
const CONSTANT_NAME:dataType = value;
```

SAMPLE CODE

BasicSampleC.fla

```
const RATE:Number = 93.39;                     ①作成した定数を
trace( "固定レートは" + RATE + "です。");     ②ここで使用して計算
var usd1:Number = 1039.99;
```

ActionScript3.0グローバル定数

定数	定義内容
Infinity : Number	正のInfinityを表す特殊な値です。
-Infinity : Number	負のInfinityを表す特殊な値です。
NaN : Number	"非数" (NaN) の値を表す Number データ型の特殊メンバーです。
undefined : *	初期化されていない型指定のない変数、または初期化されていないダイナミックオブジェクトプロパティに適用される特殊な値です。

1-3-1 演算子の種類と優先順位を理解して使う

数値演算、代入、大小判定、論理判定など、プログラミング中で行う演算は、演算子を使用して行います。演算子はオペレータ（Operator）が原語で、演算の対象はオペランド（Operand）と呼ばれます。例えば1 + 2という演算の場合、演算子は「+」で、「1」と「2」の数値リテラルがオペランドになります。ActionScriptでは、リテラル、変数、式をオペランドとして記述できます。

ActionScriptのデータ型とデフォルト値

演算子の種類	含まれる演算子
基本	[] {x:y} () f(x) new x.y x[y] <></> @ :: ..
後置	x++ x--
単項	++x --x + - ~ ! delete typeof void
乗法	* / %
加算	+ -
ビット単位シフト	<< >> >>>
関係	< > <= >= as in instanceof is
等価	== != === !==
ビット単位の論理積(AND)	&
ビット単位の排他的論理和(XOR)	^
ビット単位の論理和(OR)	|
論理積(AND)	&&
論理和(OR)	||
条件	?:
代入	= *= /= %= += -= <<= >>= >>>= &= ^= |=
カンマ	,

※ 表の上にある演算子ほど優先順位が高くなります。

SAMPLE CODE

```
var i:int = I0;
var sum:int = i++ + 5;
```

x = y + 2 * z; という演算式のように1つの式に複数の演算子が使用される場合、左から順に演算を行ってゆくと、x = の代入を行うときに = の右側の演算が行われていないことになり、演算が行えません。実際には、= の優先順位が低いため、+ と * の演算が最初に行われます。ActionScriptでは、演算子に優先順位をつけて感覚的にも矛盾が生じないプログラミングができるようになっています。サンプルコードの2行目の演算では、++ 後置演算子と+ 演算子が使用されていますが、++演算子の優先順位が高いため、sum:int = 11 + 5; と同じ処理内容になります。

1-3-2 演算子の結合規則を理解して使う

優先順位のみでは解決できない問題があります。同じ順位の演算子が1つの式で使用された場合です。例えば、x = y + f() - 1; はy + f() と、f() - 1のどちらが最初に行われるかということが決まっていないと、正しいプログラミングができません。この問題は結合規則で解決されます。ActionScriptでは、代入演算子と条件演算子については、その右側から（これを右結合と呼びます）、それ以外の演算子については左から演算が行われる（これを左結合と呼びます）ように決めており、演算の実行順序が明確に決定されます。

SAMPLE CODE

```
var i:int = 10;
var k:int = i = i + 1;
trace( k );
```

2行目で1つの演算式に = 演算子が2回使用されています。= 演算子は右結合であるため、次の順序で計算が行われます。

```
① i = i + 1
② j:int = i
```

▶ 実行結果

```
11
```

1－4 条件分岐を行う

1-4-1 条件によって異なる処理を行う

条件分岐は、プログラム実行中に値の判定を行い、処理を分岐する構文です。例えば「読み込むビデオのサイズによって表示する位置を変更する」というような処理を行う場合、条件分岐が必要になります。もっとも基本的な条件分岐は、if .. elseです。if文の構文は、ifの後ろに( )を書き、その中に条件分岐の基準となる条件（condition）を書きます。条件はtrue / falseで判定できる必要があります。elseを使用して、条件の値がfalseのときに行う処理を書くことができます。条件には、変数、式、リテラルを書くことができます。

if ... else 文の構文

```
if ( condition )
{
  // conditionがtrueの場合の処理
}
else if ( condition2 )
{
  // condition2がtrueの場合の処理
}
...   //ここに別のelse if ( ) { } を書いて条件を追加できます
else
{
  // condition2がfalseの場合の処理
}
```

※ ... の部分に複数のelse if句を記述できます。
※ else if、elseの使用はオプションです。処理が不要の場合、書く必要はありません。

SAMPLE CODE

BasicSampleD.fla

```
var i:Number = 10.1;
if ( i is int )
{
  trace( "i はint型です。: " + i );
}
else
{
  trace(" i はint型ではありません。: " + i );
}
```

1-4-2 複数の条件を効率よく処理する（switch）

if..else構文を使用して複数の条件分岐を行えますが、条件が多い場合、switchを使用したほうが便利です。switchに使用する条件式は必ずしもtrue / falseを返す必要もありません。switch文の構文は、switchの後ろに( )を書き、その中に条件分岐の基準となる条件（condition）を書きます。条件は、任意の式が使用できます。式が返す値をcase句で判定して分岐処理を行います。

switch 文の構文

```
switch ( condition )
{
  case value1:
  // conditionがvalue1を返す場合の処理
  break;

  case value2:
  // conditionがvalue2を返す場合の処理
  break;
  ...

  default:
  // conditionがすべてのcaseに該当しない場合の処理
  break;
}
```

※ defaultの使用はオプションです。処理が不要の場合、書く必要はありません。

SAMPLE CODE

シンプルなswitch文

BasicSampleE.fla

```
var season:String = "春"; ―――― ①
switch ( season )
{
  case "春":
  trace( "暖かいですね" ); ―――― ②この文が実行される
  break;
  case "夏":
  trace( "暑いですね" );
  break;
```

```
  case "秋":
  trace( "涼しいですね" );
  break;
  case "冬":
  trace( "寒いですね" );
  break;
}
```

もっともシンプルなswitch文の例です。①で宣言した変数seasonに格納されている文字列を判定して分岐します。サンプルの場合、"春"という値なので、②が実行されます。

SAMPLE CODE

breakを利用した高度なswitch文

BasicSampleF.fla

```
var score:int = 6; ―――― ①
switch ( score )
{
  case 10:
  case 9:
    trace( "とても" );
  case 8:
  case 7:
    trace( "良い点数です。" ); ―――― ②
    break; ―――― ③

  case 5:
    trace( "いちおう" );
  case 6:
    trace( "合格です。" );
    break;
  default:
    trace( "不合格です。" );
    break;
}
```

①で作成した変数scoreの値をswitch文で判定しています。10、9のときは、「とても」と表示しますが、breakがないため、そのまま②を実行して③のbreakでswitchを終了します。5のときも同様の処理を行います。このように、breakをうまく使用して高度な条件分岐処理を行うことができます。

1-5-1 条件を指定して繰り返し処理を行う(while)

while文を使用すると、ある式の値がtrueの間、処理を繰り返すことができます。while文は、繰り返しの回数が明確に決まらないときなどに使用されます。例えば「ある文章の中から指定した文字列の出現回数をカウントする」というような処理にはwhile文の使用が適しています。while文の構文は、while の後ろに ( ) を書き、その中に繰り返しの基準となる条件(condition)を書きます。条件はtrue / false で判定できる必要があり、条件がtrueの間、while以下のブロックを繰り返し実行します。

while 文の構文

```
while ( condition )
{
  // conditionがtrueの間繰り返す処理
}
```

SAMPLE CODE

文字列の出現回数を調べる

BitmapSampleG.fla

```
var s:String = "while文を使用すると、ある式の値がtrueの間、処理を繰り返すことができます。
while文は、繰り返しの回数が明確に決まらないときなどに使用されます。";
var target:String = "while"; //出現回数を調べる文字列
var start:int = 0;
var cnt:int = 0;
start = s.indexOf( target, start );
while ( start != -1 )
{
  start = s.indexOf( target, start + 1 );
  cnt++;
}
trace ( target + " は " + cnt  + " 個あります。" );
```

String.indexOf(val:String, startIndex:Number = 0):int メソッドは、varで指定した文字列を、String文字列のstartIndexで指定された文字位置から検索して見つかった位置を返します。見つからない場合、−1を返します。このメソッドを利用してtargetに設定した文字列の出現回数をカウントしています。

1-5-2 回数を指定して処理を繰り返し実行する（for）

配列の要素のように、あらかじめ個数（ループする回数）がわかる場合、for文による繰り返し処理を行います。for文では、繰り返し回数のカウンタとして利用するint型の変数を使って処理を行います。for文の構文は、for の後ろに ( ) を書き、その中に ;（コロン）で区切られた3つの部分を書きます。3つの部分には、それぞれ①繰り返し前に1回行う処理（init）、②繰り返しの継続条件（condition）、③繰り返しごとに行う処理（next）を書きます。

for 文の構文

```
for ( ①init ; ②condition ; ③next )
{
  // 繰り返し行う処理
}
```

※ 繰り返し行う処理が1行の場合、ブロックを示す{ }は省略できます。

SAMPLE CODE

シンプルな for

```
var i:int;
for ( i = 0 ; i <= 10 ; i++ ) trace( i );
```

もっともシンプルな例です。for ( ) の中で、まずループを開始する前に ① i = 0 を実行します。その後、②条件である i <= 10 を満たす（true）間、3行目から5行目のブロック内に書かれた処理を実行します。1回ループするごとに③ i++ を行うため、10回処理を繰り返すとiの値は11になり、②の条件がfalseになり、繰り返し処理は終了します。

SAMPLE CODE

配列内の要素をすべて調べる

BasicSampleH.fla

```
var pref:Array = [ "北海道", "東京都", "神奈川県", "福岡県" ];   ①String配列の要素を作成
var i:int;

for ( i = 0 ; i < pref.length ; i++ )   ②for文で①のString配列の要素をチェック
{
  switch ( pref[ i ] )   ③switch文でString値の判定
  {
    case "北海道":
```

```
      trace( pref[ i ] +  ":北海道地方です" );
      break;

      case "東京都":
      case "神奈川県":
      trace( pref[ i ] +  ":関東地方です" );
      break;

      case "福岡県":
      trace( pref[ i ] +  ":九州地方です" );
      break;
    }
}
```

④case文を連続して複数の値をまとめて処理

①で作成したString配列の要素を②のfor文を使用して1つずつチェックします。同様の処理は配列内の要素をチェックするときのテクニックとして使用できます。③ではswitch文でString値の判定を行っています。「東京都または神奈川県」というように、いくつかの値をまとめて処理するには、④で行っているようにcase文を連続して書くことで行えます。

▶ 実行結果

```
北海道:北海道地方です
東京都:関東地方です
神奈川県:関東地方です
福岡県:九州地方です
```

1-5-3 オブジェクト内のプロパティ名を取り出す(for...in)

オブジェクトに動的に追加されたプロパティ名を1つずつ順番に取得したい場合、for...inによる繰り返し処理ですべてのプロパティ名を取り出すことができます。for...in句は配列の要素取得にも使用できます。for ..in句の構文は、for の後ろに ( ) を書き、その中にプロパティ名を格納する変数(propName)と、対象のオブジェクト(targetObject)を指定します。すべてのプロパティについて繰り返し処理が行われ、ループごとに変数にプロパティ名が格納されます。

for ...in 句の構文

```
for ( var propName:String in targetObject )
{
  // 各プロパティ名ごとに行う処理
}
```

SAMPLE CODE

Objectオブジェクト内のプロパティとその値をすべて取り出す

BasicSample1.fla

```
var targetObject:Object = { name: "Miyata", gender: "Male" }; ──①テスト用のオブジェクトを作成

for ( var propertyName:String in targetObject ) ──②プロパティ名を格納する変数と対象のオブジェクトを指定
{
  trace( "プロパティ名=" + propertyName + ", 値=" + targetObject[ propertyName
  ] ); ──③プロパティの値を取り出す
}
```

①はテスト用のオブジェクトを作成しています。オブジェクト内のプロパティの数だけ繰り返し処理が行われ、②で指定したpropertyName:String変数にその名前が格納されます。③で行っているように、targetObject[propertyName]という記述でプロパティの値を取り出すことができます。

▶ 実行結果

```
プロパティ名=name, 値=Miyata
プロパティ名=gender, 値=Male
```

SAMPLE CODE

カスタムクラス内のプロパティ名と値を取り出す

CustomClass.as

```
package {
  public dynamic class CustomClass {
    public var myProperty:String="固定プロパティ";
  }
}
```

BasicSampleJ.fla

```
var targetObject:customClass = new CustomClass();
targetObject.myString="Miyata";
targetObject.myInt=1;

for (var propertyName:String in targetObject) {
  trace( "プロパティ名=" + propertyName + ", 値=" + targetObject[
  propertyName ] );
}
```

dynamic指定されたcustomClassに動的に追加されたmyString、myIntプロパティのみがfor...inの対象として出力されます。

▶ 実行結果

```
プロパティ名=myString, 値=Miyata
プロパティ名=myInt, 値=1
```

1-5-4 XMLやオブジェクト内のプロパティ値を取り出す（for each...in）

オブジェクト内のプロパティ値を1つずつ順番に取得したい場合、個数と順序の取得ができないため、for文で繰り返し処理ができません。この場合、for each...inによる繰り返し処理ですべてのプロパティ名を取り出すことができます。for each...in句はXML処理の機能として導入されましたが、XMLのみでなく、配列の要素取得にも使用できます。for each...in句の構文は、for each の後ろに ( ) を書き、その中にプロパティ値を格納する変数（propValue）と、対象のオブジェクト（targetObject）を指定します。すべてのプロパティについて繰り返し処理が行われ、ループごとに変数にプロパティ値が格納されます。

for each...in 句の構文

```
for each ( var propName:dataType in targetObject )
{
  // 各プロパティ値ごとに行う処理
}
```

SAMPLE CODE

Objectオブジェクト内のプロパティ値をすべて取り出す　BasicSampleK.fla

```
var targetObject:Object = { name: "Miyata", gender: "Male" };   ①テスト用のオブジェクトを作成

for each ( var propertyValue:String in targetObject )   ②プロパティ値を格納する変数と対象のオブジェクトを指定
{
  trace( "プロパティ値=" + propertyValue );   ③プロパティの値を取り出す
}
```

①はテスト用のオブジェクトを作成しています。オブジェクト内のプロパティの数だけ繰り返し処理が行われ、②で指定したpropertyValue変数にその値が格納されます。③でプロパティの値をtrace出力します。

▶ 実行結果

```
プロパティ値=Male
プロパティ値=Miyata
```

PART
1
2
3
4
5
6
7
8
9

SAMPLE CODE

XML内の繰り返し項目を1つずつ取り出す

BasicSampleL.fla

```
var dataXML:XML =
<products>
<productName>Dell</productName>
<productName>HP</productName>
</products>;

for each (var product in dataXML.productName)
{
  trace( product );
}
```

XMLリテラルで作成されたdataXMLの繰り返し要素であるproductNameを処理します。

▶ 実行結果

```
Dell
HP
```

1-6-1 関数を作成する

関数は、繰り返し使用できるコードブロックです。ActionScriptプログラムで複数の個所で同じ処理を行うような場合、その部分を関数にしておくと効率が良くなります。ActionScriptでは、クラス内に関数を定義した場合、メソッド（method）と呼び、それ以外の場所で定義された関数はそのまま関数（function）と呼びます。関数を作成するには、functionキーワードに続けて関数名を書きます。関数名の後ろに ( ) を書き、その中にその関数が受けるパラメータ（param1,...）を書きます。パラメータを受けない場合、( ) 内は空白にします。関数の戻り値がある場合、: に続けてそのデータ型（returnType）を書きます。戻り値がない場合、: voidと記述します。

関数の構文

```
function functionName ( [param1:dataType,...paramN:dataType] ) :
returnType
{
  // 関数の処理
}
```

※ [ ] は必要がない場合、省略可能なことを意味します。
※ param1:dataType,...paramN:dataTypeは1つ以上のパラメータが記述できることを意味します。

SAMPLE CODE

シンプルな関数の作成と呼び出し　BasicSampleM.fla

```
function simpleFunction( ):void {
  trace( "simpleFunction()が実行されました。" );     ①
}
simpleFunction();                                    ②
```

①で作成しているsimpleFunction関数はパラメータを取らず、戻り値を返しません。②で関数の呼び出しをしています。

▶ 実行結果

```
simpleFunction()が実行されました。
```

1-6-2 関数にパラメータを渡す

関数はパラメータを受けることで、固定的な処理だけでなく動的な機能を果たすことができます。定義した関数のパラメータは、その関数内でのみ有効な変数として機能します。1-6-1「関数を作成する」で解説したように、関数が受けるパラメータは関数名の後ろに()を書き、その中に受けるパラメータ(param1,...)を書きます。パラメータを受けない場合、()内は空白にします。

関数パラメータの構文

```
function functionName ( [param1:dataType,...] ) : returnType
{
  // param1,... を変数として使用します
}
```

※ [ ] は必要がない場合、省略可能なことを意味します。
※ param1:dataType,...paramN:dataTypeは1つ以上のパラメータが記述できることを意味します。

SAMPLE CODE

関数にパラメータを渡す

BasicSampleN.fla

```
function getWordCount( sentence:String, targetWord:String ):void { ―①
  var start:int=0;
  var cnt:int=0;
  start = sentence.indexOf( targetWord, start );
  while ( start != -1 ) {
    start=sentence.indexOf( targetWord, start+1);
    cnt++;
  }
  trace( targetWord + " は " + cnt  + " 個あります。" );
}
var sentence:String =  "while文を使用すると、ある式の値がtrueの間、処理を繰り返すことができます。while文は、繰り返しの回数が明確に決まらないときなどに使用されます。";
getWordCount( sentence, "while" ); ―②
```

①で作成しているgetWordCount関数は2つのStringパラメータを取ります。関数の処理は、1番目のパラメータ(sentence)で与えられるStringから2番目のパラメータ(targetWord)で与えられるStringを検索してその個数をカウントします。②でgetWordCount関数の呼び出しを行っています。

1-6-3 関数からの戻り値を利用する

関数は呼び出し元に値を返すことができます。呼び出し元では、返された値は変数に代入するなどして利用できます 1-6-2「関数にパラメータを渡す」で解説したように、関数の戻り値がある場合、() の後ろの : に続けてそのデータ型（dataType）を書きます。戻り値がない場合、: void（ボイドと読みます）を記述します。

戻り値を定義した場合、関数内で必ずreturnを使用して指定したデータ型の値を返します。

関数戻り値の構文

```
function functionName ( [param1:dataType,...] ) : returnType
{
  // 戻り値を定義した場合、必ずreturn句で指定したデータ型を返します
  return returnData;
}
```

SAMPLE CODE

誤った関数定義

```
function wrongFunction( param:int ):Boolean {
  var returnValue:Boolean;
  if (param<0) {
    return false;
  }
  if (param>0) {
    return true;
  }
}
```

定義した関数wrongFunctionはBoolean型の戻り値を返すと定義していますが、paramの値が0のとき、何もしないで終了します。つまり戻り値を戻しません。そのため、この関数はコンパイルエラーを発生します。

SAMPLE CODE

関数からの戻り値を利用する BasicSample0.fla

```
function getWordCount( sentence:String, targetWord:String ):int {―①
  var start:int=0;
  var cnt:int=0;
  start = sentence.indexOf( targetWord, start );
  while ( start != -1 ) {
    start=sentence.indexOf( targetWord, start+1);
    cnt++;
  }
  return cnt;
}
var sentence:String = "while文を使用すると、ある式の値がtrueの間、処理を繰り返すことができます。while文は、繰り返しの回数が明確に決まらないときなどに使用されます。";
var cnt:int = getWordCount( sentence, "while" );――――――②
trace( "while は " + cnt  + " 個あります。" );
```

1-6-2「関数にパラメータを渡す」の関数の改良型です。①で作成しているgetWordCount関数は2つのStringパラメータを取り、1番目のパラメータ（sentence）で与えられるStringから2番目のパラメータ（targetWord）で与えられるStringを検索してその個数をカウントし、カウント値を戻り値として返します。

②でgetWordCount関数の呼び出しを行い、戻り値をcnt変数に代入しています。

▶ 実行結果

```
while は 2 個あります。
```

1-6-4 関数のパラメータ数を可変にする (... rest)

ActionScriptでは、関数が受けるパラメータの数を可変にすることができます。この場合、渡されたパラメータは配列として処理できるようになります。つまり、いくつのパラメータが渡されているかをlengthプロパティで調べることができます。関数のパラメータ数を可変にするには、関数名に続く ( ) 内で...restを記述します。この場合、restは関数内で配列としてパラメータを取り出すために使用できます。必ずしもrestという名前を使用する必要はありません。

可変パラメータの構文

```
function functionName ( ... rest ) : returnType
{
  // restは配列として、渡されたパラメータを格納しています
}
```

SAMPLE CODE

関数からの戻り値を利用する

BasicSampleP.fla

```
function getSum(... params):Number {          ①
  var sum:Number=0;
  for (var i:uint = 0; i < params.length; i++) {
    sum=sum+params[i];                           ②
  }
  return sum;
}
var sum5:Number=getSum(1,2,3,4,5);
var sum10:Number=getSum(1,2,3,4,5,6,7,8,0,10);
trace( "1から5までの和は" + sum5 + ", 1から10までの和は" + sum10 + "です。");
```

①からの関数getSumは、可変個の引数をとり、渡された引数の和を返します。paramsという配列に渡されたパラメータが格納されます。その処理を②で処理して変数sumに合計値を格納しています。

▶ 実行結果

```
1から5までの和は15, 1から10までの和は46です。
```

1-7 エラー処理を行う

1-7-1 エラーが発生したときの処理を行う

ActionScriptプログラムの実行中にエラーが発生すると、Flash Playerは例外をスロー(throw)してアプリケーションの実行を中断します。スローされた例外はキャッチ(catch:捕捉)してプログラムで適切に処理することができます。この処理をエラー処理、あるいは例外処理と呼びます。Flash Playerは、例外をスローする際にErrorオブジェクトを生成してエラーの詳細情報を格納します。キャッチしたコードでそのErrorオブジェクトの内容を見て処理を行うことができます。例外を発生する可能性があるコード部分はtry { } ブロックに含めます。ブロックの後にcatch (error: ErrorType) { } を記述し、そのブロックでエラー処理を行います。catchブロックはエラーの種類ごとに複数書くことができます。finally { } ブロックにより、エラーが発生したときもそうでないときにも実行するコードを記述できます。

エラー処理の構文

```
try
{
  //エラーが発生する可能性のあるコード
}
catch( error[:ErrorType1] )
{
  //ErrorType1が補足された場合のエラー処理
  //ErrorType1指定しない場合はすべてのエラーが補足される
)
[catch( error:ErrorType2 )
{
  //ErrorType2が補足された場合のエラー処理
}]
[finally
{
  //必ず実行する処理
}]
```

※ [ ] は必要がない場合、省略可能なことを意味します。つまり、catchブロックは最低1つ以上記述します。finallyブロックは必要なら記述します。

SAMPLE CODE

try - catchによるエラー処理

BasicSampleN.fla

```
import flash.display.MovieClip;
var mc:MovieClip ———————①MovieClipクラスの変数を宣言
try {
```

```
  mc.play(); ―――――――――――――――――――――②メソッドを呼び出す
} catch( e:TypeError ) { ―――――――――――③catchブロックの処理
    trace( e );
} finally {
    trace( "finallyが呼ばれました" );
}
```

①でMovieClipクラスの変数を宣言していますが、newによるインスタンス化を行っておらず、null値を持っています。そのため、②でメソッドを呼び出すときにランタイムエラーが発生します。このときのErrorオブジェクトはTypeErrorなので、③から始まるcatchブロックで処理された後にfinallyブロックが実行されます。

▶ 実行結果

```
TypeError: Error #1009: null のオブジェクト参照のプロパティまたはメソッドにアクセスすることはできません。
finallyが呼ばれました
```

Errorオブジェクトの種類

Errorオブジェクト	発生タイミング
ArgumentError	関数で指定された引数がその関数の定義に適合していないとき
DefinitionError	定義済みの識別子をユーザーコードで定義しようとしたとき
EvalError	ユーザーコードがFunctionオブジェクトでeval() 関数を呼び出す、またはnew演算子を使用しようとしたとき
IllegalOperationError	メソッドが実装されていないか、実装が対応していないとき
InvalidSWFError	破損したSWFファイルを検出したとき
IOError	入出力エラーが発生したとき
MemoryError	メモリ割り当て要求が失敗したとき
RangeError	数値の値が許容範囲内に収まらないとき
ReferenceError	sealed指定された動的でないオブジェクトに対して未定義プロパティを参照しようとしたとき
ScriptTimeoutError	スクリプトのタイムアウト間隔が経過したとき
SecurityError	セキュリティの侵害が発生したとき
StackOverflowError	スクリプトで使用できるスタックを使い切ったとき
SyntaxError	解析エラーが発生したとき
TypeError	オペランドの予期される型と実際の型が異なるとき
URIError	グローバルなURI処理関数のいずれかが定義に合わない方法で使用されたとき
VerifyError	不正な形式、または破損したSWFファイルが検出されたとき
VideoError	FLVPlaybackまたはVideoPlayerクラスで問題が発生したとき

1-8-1 イベントリスナーを作成する

ユーザがマウスをクリックするとclickイベントが発生し、MouseEventというオブジェクトが生成され、登録された関数でそのイベントとオブジェクトを捕捉して処理できます。このようにユーザの操作やシステムから発せられる情報はイベントとして送信され、登録された関数で処理を行います。その関数はイベントの発生を監視することになるため、イベントリスナー(Event Listener)関数、またはイベントハンドラ(Event Handler)と呼ばれます。通常のActionScript関数またはクラス内に作成されたメソッドをイベントリスナーとして使用できます。そのため、イベントリスナーの作成方法は関数の作成方法と共通します。リスナー関数はイベントオブジェクトをパラメータ(event:eventObjectClass)として受けるところが特徴的です。パラメータのデータ型は処理するイベントオブジェクトを指定します。例えばマウス関連のイベントの場合、MouseEventオブジェクトが送出されるので、event:MouseEventとします。

イベントリスナーの構文

```
function listener( event:eventObjectClass ) : void
{
  // eventパラメータを使用したイベントの処理
}
```

※ パラメータ名は必ずしもeventである必要はありません。

イベントリスナーの登録

イベントリスナー関数の登録は、addEventListener()関数を使用します。この関数の第1パラメータ(type)に対象のイベント名、第2パラメータ(listener)にリスナー関数の名前を渡して実行することで登録が完了します。このメソッドはEventDispatcherクラスのものですが、このクラスはDisplayObjectのスーパークラスであるため、ほとんどのコンポーネントでaddEventListener()メソッドを使用できます。

イベントリスナー登録のメソッド

```
public function addEventListener(type:String, listener:Function,
useCapture:Boolean = false, priority:int = 0, useWeakReference:
Boolean = false):void
```

※ 第3パラメータ以下はデフォルト値が設定済みで、省略可能です。

イベントリスナーの削除

イベントリスナー関数の登録を削除できます。削除はremoveEventListener()関数を使用し

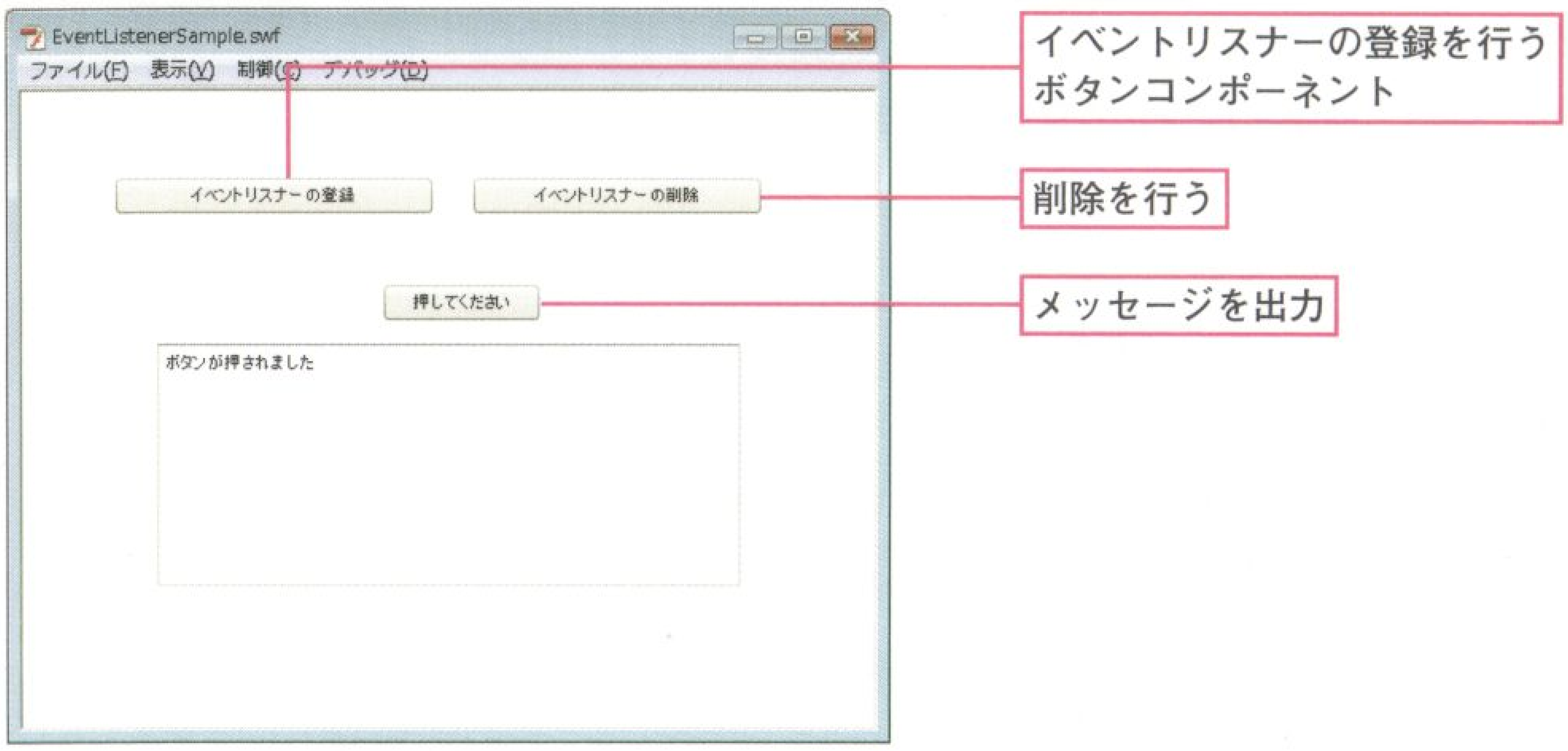
ます。この関数の第1パラメータ(type)に対象のイベント名、第2パラメータ(listener)にリスナー関数の名前を渡して実行することで削除が完了します。このメソッドもEventDispatcherクラスのものです。

イベントリスナー削除のメソッド

```
public function removeEventListener(type:String, listener:Function,
useCapture:Boolean = false):void
```

※ 第3パラメータはデフォルト値が設定済みで、省略可能です。

SAMPLE



SAMPLE CODE

イベントリスナーの登録/削除処理

BasicSampleR.fla

```
//イベントリスナーの登録ボタン
this.button1.addEventListener( MouseEvent.CLICK, button1Listener );
function button1Listener( event:MouseEvent ):void {
  this.button3.addEventListener( MouseEvent.CLICK, button3Listener );
}
//イベントリスナーの解除ボタン
this.button2.addEventListener( MouseEvent.CLICK, button2Listener );
function button2Listener( event:MouseEvent ):void {
  this.button3.removeEventListener( MouseEvent.CLICK, button3Listener
);
}
//メッセージを出力する処理
function button3Listener( e:MouseEvent ):void {
  this.output_ta.text += "ボタンが押されました";
}
```

1-8-2 イベントオブジェクトを利用する

PART 1 2 3 4 5 6 7 8 9

1-8-1「イベントリスナーを作成する」で解説したように、イベントの種類に応じて生成されるオブジェクトが異なります。例えばマウス関連の操作ではMouseEventオブジェクトが、ファイルをロードするときのprogressイベントではProgressEventオブジェクトが生成されます。このようにイベントの種類によってオブジェクトが異なるのは、格納する情報が異なるためです。リスナーの処理ではこのイベントオブジェクトが持つ情報を利用します。例として、MouseEventオブジェクトが持つプロパティを見てみると、下表のようなプロパティが用意されています。例えばボタンがクリックされたとき、CtrlキーとAltキーが押されている場合、altKeyプロパティとctrlKeyプロパティの値がtrueにセットされます。targetプロパティはユーザから見た場合、通常イベントの発生元への参照を表します。

MouseEventオブジェクトの主なプロパティ

プロパティ	内容
altKey:Boolean	Altキーがアクティブの場合、true
buttonDown:Boolean	マウスの主ボタンが押されている場合、true
ctrlKey:Boolean	Ctrlキーがアクティブの場合、true
localX:Number	スプライトを基準としたマウスカーソル水平位置
localY:Number	スプライトを基準としたマウスカーソル垂直位置
shiftKey:Boolean	Shiftキーがアクティブの場合、true
stageX:Number	ステージを基準としてマウスカーソル水平位置
stageY:Number	ステージを基準としてマウスカーソル垂直位置
target:Object	イベントターゲット
type:String	イベントのタイプ

1-8-1「イベントリスナーを作成する」のサンプルを拡張してイベントオブジェクトの内容を出力します。

マウス関連イベントの定数

定数	イベント
CLICK	click
DOUBLE_CLICK	doubleClick
MOUSE_DOWN	mouseDown

マウス関連イベントの定数（続き）

定数	イベント
MOUSE_MOVE	mouseMove
MOUSE_OUT	mouseOut
MOUSE_OVER	mouseOver
MOUSE_UP	mouseUp
MOUSE_WHEEL	mouseWheel
ROLL_OUT	rollOut
ROLL_OVER	rollOver

SAMPLE



SAMPLE CODE

イベントオブジェクトの利用

BasicSampleS.fla

```
//イベントリスナーの登録ボタン                                ①clickイベントリスナーの登録
this.button1.addEventListener( MouseEvent.CLICK, button1Listener );
function button1Listener( event:MouseEvent ):void {
  this.button3.addEventListener( MouseEvent.CLICK, button3Listener );
  this.button3.addEventListener( MouseEvent.ROLL_OVER, button3
  Listener );                                                 ②rolloverイベントリスナーの登録
}
//イベントリスナーの解除ボタン                                ③リスナーの削除
this.button2.addEventListener( MouseEvent.CLICK, button2Listener );
function button2Listener( event:MouseEvent ):void {
  this.button3.removeEventListener( MouseEvent.CLICK, button3
  Listener );
```

PART 1

```
  this.button3.removeEventListener( MouseEvent.ROLL_OVER, button3
  Listener );
}
//メッセージを出力する処理
function button3Listener( e:MouseEvent ):void {
  this.output_ta.text +=
    イベントの種類:  + e.type + "¥n" +
    Altキーのアクティブ: + e.altKey + "¥n" +
    ctrlキーのアクティブ: + e.ctrlKey + "¥n" +
    shiftキーのアクティブ: + e.shiftKey + "¥n" +
    ローカル座標:( + e.localX + "," + e.localY + ")" + "¥n" +
    ステージ座標:( + e.stageX + "," + e.stageY + ")" + "¥n";
}
```

③リスナーの削除

①と②で"click"と"rollover"イベントに対してリスナーの登録を行っています。この場合、③で行っているように、リスナーの削除も2つのイベントに対して行う必要があります。

COLUMN

イベントの発生とイベントフロー

Flash PlayerまたはAIRでは、イベントが発生するたびにイベントオブジェクトが作成され、送出されます。イベントターゲットが表示リスト内にない場合、イベントターゲットに対して直接にイベントオブジェクトが送出されます。この時、イベントターゲットが表示リスト内にあればイベントオブジェクトは表示リストに対して送出され、表示リストの階層内を経由してイベントターゲットに到達します。表示リスト内をイベントオブジェクトが伝わる経路がイベントフローです。表示リストの階層構造の最上位はステージですが、イベントオブジェクトが作成されて送出されると、そのオブジェクトはステージからターゲットノードまでを往復します。

イベントフローの第1段階はキャプチャ段階と呼ばれ、ステージからターゲットノードの親までのノードまで伝播(でんぱん)します。次の第2段階は、ターゲット段階と呼ばれ、ターゲットノードに到達します。最後の第3段階はバブリング段階と呼ばれ、ターゲットノードの親からステージにオブジェクトが戻されます。

1－9 クラスとインスタンス

1-9-1 クラスを宣言する

クラスを使わなくてもFlashのActionScriptプログラミングが可能ですが、作成するアプリケーションの規模が大きい場合やチームで開発する場合などはオブジェクト指向の方法を取り入れると開発の効率性や保守性が高まります。また、ActionScript 3.0コンポーネントリファレンスガイドで示される開発者向けのプログラミングインタフェースはクラスで構成されるライブラリとなっています。クラスの宣言は、classキーワードに続けてクラス名を記述します。{ }ブロック内にクラスのプロパティとなる変数と、メソッドとなる関数を宣言します。

クラスの宣言構文

```
[package packageName { ]
[dynamic] [public | internal] [final] class className [ extends
superClass ] [ implements interfaceName[, interfaceName... ] ] {
        //クラスのプロパティとメソッドを作成します
}
[ } ] //package宣言のブロック終了
```

※ [ ]で囲まれたキーワードはオプションで、必要ない場合、省略可能です。次の表にオプションの概要を示します。

class宣言のときに追加できるオプション

キーワード	概要
package packageName	packageNameで示されるパッケージに属するクラスにします。パッケージ名はクラスファイルを格納するフォルダ名と同一にします。
dynamic	クラスのインスタンスを生成後にプロパティを追加できるようにします。追加されたプロパティはfor in 構文で取り出すことが可能になります。
public	このクラスを使用する場所に制限を設けません。パッケージ内外にかかわらず、使用可能にします。
internal	同じパッケージ内の他クラスからのみ、このクラスを使用できるようにします。public/internalを指定しない場合のデフォルトはinternalです。
final	このクラスを拡張できないようにします。
extends superClass	superClassで指定されるクラスを拡張するクラスを宣言します。
implements interfaceName	interfaceNameで指定されるインタフェースを実装するクラスを宣言します。

※ publicとinternalは、宣言するクラスが外部のどの場所から使用できるかを制限するアクセス制御の働きになります。

1-9-2 クラスを使う

クラスはインスタンスを生成して初めてオブジェクト(実体)となり、Flash Player上で動作するようになります。クラスのインスタンスを生成するには、変数を宣言してnew演算子の結果を代入します。

PART 1 2 3 4 5 6 7 8 9

クラスインスタンス作成の構文

```
var instanceName: className = new constructor( parameters );
```

※ classNameとconstructorは同じ名前になります。

SAMPLE CODE

もっともシンプルなクラス宣言と使用(クラス宣言)

SimpleClass.as

```
package { ——①パッケージの宣言
  public class SimpleClass { ——②SimpleClassクラスをpublicで作成
    public var myProperty:String; ——③クラスのプロパティは変数として宣言
    public function myMethod():void { ——④クラスのメソッドは関数として宣言
      trace( "myProperty=" + myProperty );
    }
  }
}
```

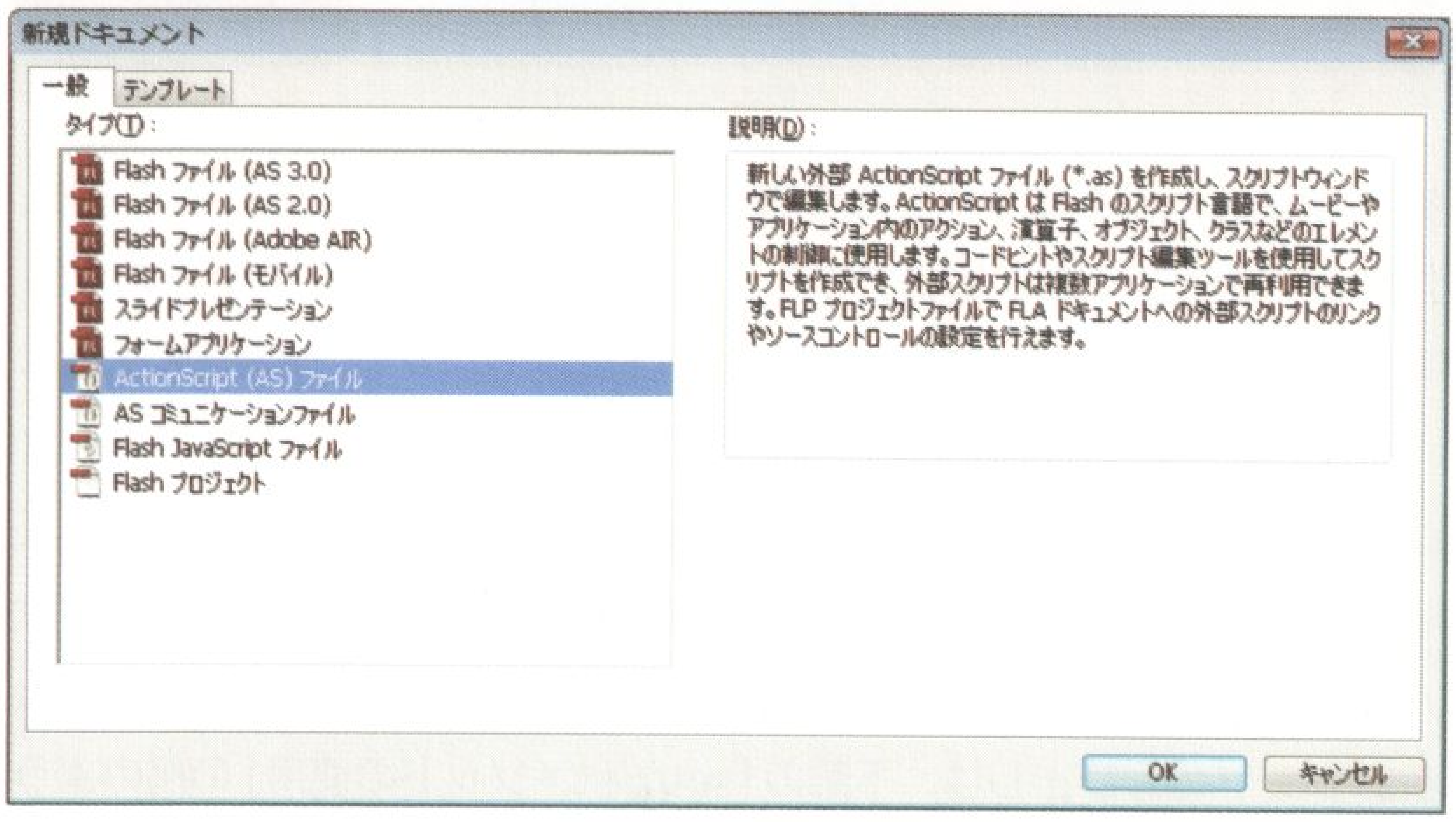
①でパッケージ宣言をします。パッケージを作成しない場合、このようにパッケージ名を書かずに宣言できます。その場合、特にフォルダを作成してasファイルを保存する必要はありません。

クラスを作成するには、タイムラインスクリプトではなく、外部ASファイルを作成する必要があります。サンプルコードはSimpleClass.asというファイル名で保存します。

コンストラクタ

コンストラクタ(constructor)関数は、new演算子によってクラスのインスタンスを作成すると同時に実行されるメソッドと言えます。コンストラクタは定義されたクラスと同じ名前になり、戻り値を指定することはできません。

SAMPLE



SAMPLE CODE

クラスを使う

BasicSampleT.fla

```
var myObj:SimpleClass = new SimpleClass(); ⑤クラスインスタンスを生成
myObj.myProperty = "プロパティの値"; ⑥クラスのプロパティに値を設定
myObj.myMethod(); ⑦クラスのメソッドを呼び出す
```

⑤でクラスインスタンスを生成します。この後インスタンスはmyObjという変数で処理できます。

サンプルコードはFLAファイルのタイムラインスクリプトに記述できます。SimpleClassクラスをFLAファイルから使用する場合、SimpleClass.asファイルを同じフォルダに保存しておきます。

▶ 実行結果

```
myProperty=プロパティの値
```

1-9-3 クラスのプロパティとメソッドを作成する

PART
1
2
3
4
5
6
7
8
9

1-9-2「クラスを使う」で解説したようにすべてのプロパティとメソッドをpublicにすると、クラスを使う側からは自由にアクセスできて便利です。しかし自由なアクセスを許可すると、不正なプロパティ値が設定されることを防ぐことができません。そのため、プロパティはprivateにしてアクセスを制限し、専用のpublicメソッドを用意して値のチェックなどを行います。クラスのprivateプロパティに値をセットするためのpublicメソッドは、setを付加して作成します。これは、setメソッドと呼ばれます。クラスのprivateプロパティ値を取り出すためのpublicメソッドは、getを付加して作成します。これは、getメソッドと呼ばれます。set / getが付加されたpublicメソッドの呼び出しは、下記の「set/getメソッドの使用」の例のように、プロパティにアクセスするように行うことができます。

setメソッドの構文

```
public function set property(newValue:dataType) : void{
        // newValueの値をチェックして代入を行います。
}
```

getメソッドの構文

```
public function get property():dataType {
        //値をreturnします。
}
```

set / getメソッドの使用

```
var myObj:MyClass = new MyClass();
myObj.property = newValue; //このとき、setメソッドが呼び出されます。
trace( myObj.property );   //このとき、getメソッドが呼び出されます。
```

※ myObj.propertyと記述していますが、propertyはpublicメソッド名です。

SAMPLE CODE

クラスのプロパティとメソッドを作成する

SimpleClass2.as

```
package {
  public class SimpleClass2 {
    private var myPrice:int = 0;                ——①クラスのプロパティを宣言
```

```
    public function set price( newValue:int ):void {
      if ( newValue < 0 ) return;  //newValueがマイナス値のとき、何もしません
      myPrice = newValue;                              ②クラスのsetメソッドを宣言
    }
    public function get price():int {  ③クラスのgetメソッドを宣言
      return myPrice;
    }
  }
}
```


①：クラスのプロパティをprivate変数として宣言します。初期値は明示的に0とします。

②：クラスのsetメソッドをpriceという名前で宣言します。このメソッドで値チェックを行い、マイナス値は無視します。

③：クラスのgetメソッドをpriceという名前で宣言します。

SAMPLE CODE

クラスのプロパティを使う

BasicSampleU.fla

```
var myObj:SimpleClass2 = new SimpleClass2();
myObj.price = -10;  ④クラスのプロパティに不正な値を設定
trace( "-10をセットしました。実際の値は" + myObj.price );
myObj.price = 10;  ⑤クラスのプロパティに正しい値を設定
trace( "10をセットしました。実際の値は" + myObj.price );
```

▶ 実行結果

```
-10をセットしました。実際の値は0
10をセットしました。実際の値は10
```

1-9-4 クラスを拡張して新しいクラスを作成する

PART 1 2 3 4 5 6 7 8 9

すでに作成済みのクラスに機能追加を行い新しいクラスを作成できることができます。この機能追加を継承(inheritance)、または拡張(extention)と呼びます。継承したクラスをさらに継承することもできるため、既存の資産を継続して利用できることになります。元のクラスをスーパークラス(super class)、拡張した新しいクラスはサブクラス(sub class)と呼ばれます。既存のActionScriptコンポーネントクラスを拡張することも可能です。クラスの拡張はextendsキーワードを使用して行います。新しいクラスにはスーパークラスの内容がすべて含まれます。

クラスの拡張構文

```
class NewClassName extends SuperClassName {}
```

SAMPLE CODE

MovieClip クラスを拡張する

SimpleClass4.as

```
package {
  import flash.display.MovieClip;
  import flash.text.TextField;

  //MovieClipクラスを拡張してSimpleClass4を作成します
  public class SimpleClass4 extends MovieClip { ―――― ①
    var outputTa:TextField;

    //このメソッドはTextFieldを作成して配置します
    public function createTextField():void {   ┐
      outputTa = new TextField();               │
      outputTa.width=stage.stageWidth*0.5;      │
      outputTa.height=stage.stageHeight*0.5;    ├― ②
      outputTa.background=true;                 │
      outputTa..backgroundColor=0xEEEEEE;       │
      addChild(outputTa);                       ┘
    }

    //このメソッドはTextFieldにMovieClipが持つプロパティ値を表示します
    public function showProperties():void {
```

```
        outputTa.appendText( getProperties() );
    }
    private function getProperties():String { ——③
        var str:String = ""
          + "currentFrame: " + currentFrame + "¥n"
          + "currentScene: " + currentScene + "¥n"
          + "framesLoaded: " + framesLoaded + "¥n"
          + "totalFrames: " + totalFrames + "¥n";
        return str;
    }
  }
}
```

ステージ上のムービークリップシンボルインスタンスはMovieClipクラスのインスタンスです。このため、ActionScriptでMovieClipクラスを拡張して新たなクラスを作成し、シンボルインスタンスと同じようにステージ上に配置できます。

①：MovieClipクラスを拡張するSimpleClass4の宣言を行っています。

②で定義されているcreateTextFieldメソッドを実行することにより、TextFieldを作成して配置します。

③から定義されているgetPropertiesメソッドはスーパークラスであるMovieClipクラスのプロパティを参照してその値を返します。

SAMPLE CODE

MovieClipを継承したSimpleClass4クラスを使う BasicSampleV.fla

```
//クラスのインスタンスを生成してステージ上に配置します。
var obj:SimpleClass4 = new SimpleClass4(); ┐
addChild( obj );                           ┘——④
//インスタンスのメソッドを呼び出します。
obj.createTextField();
obj.showProperties();
```

④でSimpleClass4のインスタンスを作成し、配置します。配置した後でインスタンスのメソッドを呼び出してTextFieldの構築とデータの表示を行います。

PART
1
2
3
4
5
6
7
8
9

1-9-5 パッケージを理解して使う

作成したクラスが多くなる場合、その整理にはパッケージによる分類が必要になります。ActionScriptコンポーネントライブラリもすべてパッケージ化されています。ActionScriptでは、パッケージ名は物理的なフォルダ名と同一です。つまり、作成したasファイルをパッケージ名と同じフォルダに格納します。パッケージ構造にすることで、同じ名前のクラスを異なるパッケージに複数作成することもできます。通常は、クラスの宣言を行うときにパッケージの宣言も行います。宣言したクラスはそのパッケージに属するものとして扱われます。

パッケージの宣言構文

```
package packageName {
  class ClassName {
    //クラス内部の定義
  }
}
```

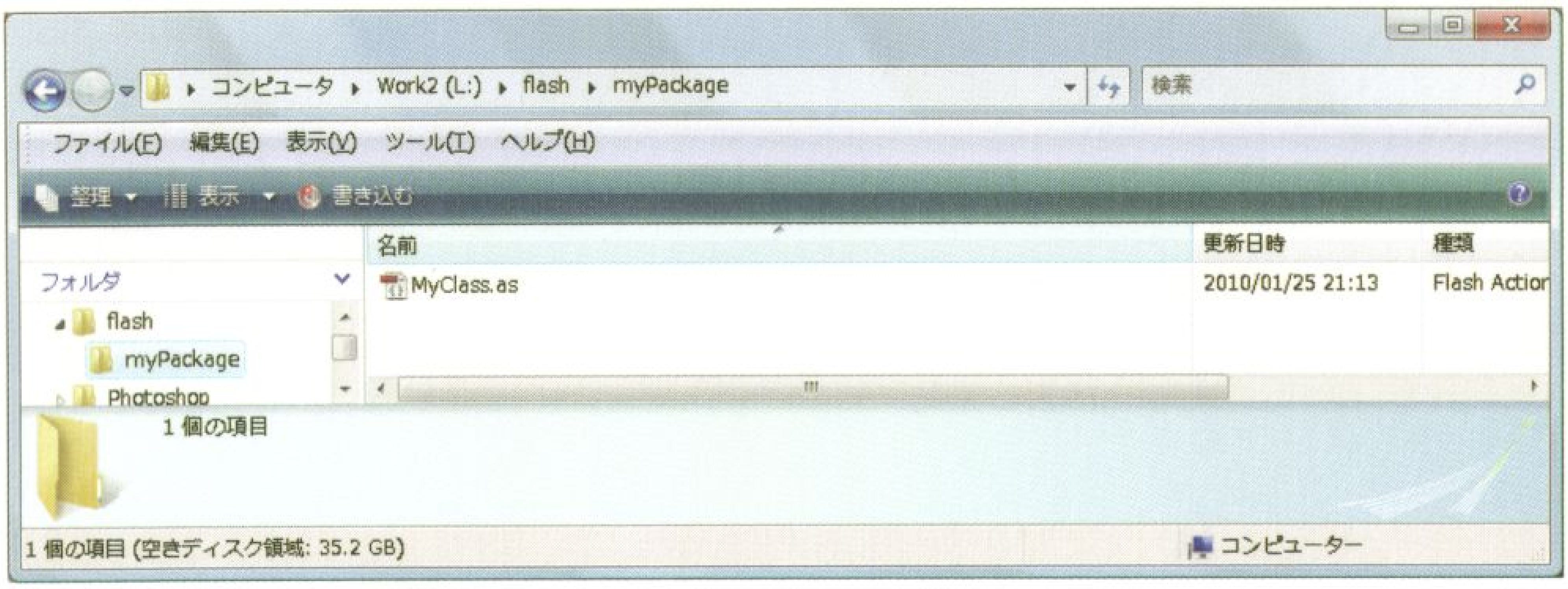
SAMPLE CODE

パッケージを使う

MyClass.as

```
package myPackage{
  dynamic public class MyClass {
    private var myProperty:String;
    public function MyClass( myPropertyValue: String ):void {
      myProperty=myPropertyValue;
    }
    public function get myproperty():String {
      return myProperty;
    }
    public function set myproperty( myPropertyValue:String ):void {
      myProperty=myPropertyValue;
    }
  }
}
```

myPackageというパッケージに属するクラスとしてMyClassを宣言します。このActionScriptはMyClass.asというファイル名でmyPackageという名前のフォルダ内に保存します。



SAMPLE CODE

パッケージ宣言したMyClassクラスを使う

PackageSample.fla

```
import myPackage.MyClass;
var myObj:MyClass = new MyClass( "パラメータ値" );
trace( myObj.myproperty );
```

myPackageパッケージのMyClassクラスを利用するアプリケーションでは、上記のようにimport文を使用してクラスの位置を明示します。

このアプリケーションファイルはmyPackageフォルダがある場所に作成します。



PART 2

日付・時間処理

2－1 日付と時刻

2-1-1 Dateオブジェクトを作成する

PART
1
2
3
4
5
6
7
8
9

ActionScriptで日付と時刻を操作するには、Dateクラスを使用します。Dateクラスのインスタンスは、例えば「2010/11/1 13:00」というような1つの日時データを保持します。日時の情報は、内部的には、1970年1月1日0時0分0秒からの経過時間をミリ秒単位に換算した値で保持されます。また、特定の日時を保持するDateインスタンスを生成するには、コンストラクタに日時に関するデータを渡します。Dateクラスのインスタンスを何も指定しないで作成すると、現在の日時を持つようになります。Dateクラスのインスタンスを作成する場合、下表のような複数の方法を使い分けて目的の日時を格納します。

Date クラスコンストラクタ

```
public function Date(yearOrTimevalue:Object, month:Number, date:
Number = 1, hour:Number = 0, minute:Number = 0, second:Number = 0,
millisecond:Number = 0)
```

■ Date クラスのコンストラクタでインスタンスを作成する方法

No.	方法	構文
1	文字列で日時を指定する	var dt:Date = new Date("文字列の日時表現"); "文字列の日時表現"には、"1/30/2014"のような文字列で日付を指定できます。
2	1970年1月1日午前0時0分0.0秒からの経過時間をミリ秒で指定する	var dt:Date = new Date(1970/1/1からのミリ秒数値); 1970/1/1からのミリ秒数値が計算されて、その時点の日時がインスタンスに格納されます。
3	年、月、日、時間、秒、ミリ秒を指定する	var dt:Date = new Date(year, month [, date, hour, minute, second, msec]); 年、月以外はデフォルト値を持つため、年と月を指定することで日時データを作成できます。この場合、第1パラメータは年の指定になります。
4	何も指定しない	var dt:Date = new Date(); コンストラクタに何もパラメータを渡さないと現在日時を作成できます。

SAMPLE CODE

日時を設定する

DateSampleA.fla

```
//特定の日時をセットする
```

```
//Stringでセット( 2015年1月1日 )
var dt1:Date = new Date( "01/01/2015" ); ―――①
trace( dt1 );

//1970/1/1 00:00からの経過ミリ秒で指定
var dt2:Date = new Date( 1300000000000 );
trace( dt2 );

//年、月、日、時、分、秒をそれぞれ指定(2012/11/31 23:59:59)
var dt3:Date = new Date( 2012, 11, 31, 23, 59, 59 );
trace( dt3 );

//年と月のみ指定(2013年1月)
var dt4:Date = new Date( 2013, 1 );
trace( dt4 );
```


Dateクラスのコンストラクタに日付のパラメータを渡してインスタンスが持つ日時を設定できます。①のように日まで指定した場合、Dateインスタンスが持つ時刻は0時0分になります。

▶ 実行結果

```
Thu Jan 1 00:00:00 GMT+0900 2015
Sun Mar 13 16:06:40 GMT+0900 2011
Mon Dec 31 23:59:59 GMT+0900 2012
Fri Feb 1 00:00:00 GMT+0900 2013
```

COLUMN

Dateクラスの toString() 書式

Dateクラスオブジェクトをtrace出力するとtoString()メソッドが自動的に呼び出されます。このときの日付フォーマットは次のようになります。

Thu Jan 1 00:00:00 GMT+0900 2015
① ② ③ ④ ⑤ ⑥

①曜日 ②月 ③日 ④時刻 ⑤標準時刻との差：日本の場合、GMT+9時間 ⑥西暦年

2-1-2 現在の日付と時刻を取得する

Dateクラスのインスタンスを作成するときに、コンストラクタにパラメータを渡さないで作成すると、その時点でクライアントPCが持つ日時がDateオブジェクトに格納されます。格納される単位はミリ秒までの詳細なものです。Dateクラスのプロパティにより、年からミリ秒までの値を取り出すことできます。保持している日時から必要な値を取り出すために、下表のようなDateクラスのプロパティを使用できます。

Dateクラスから日時を取得するためのプロパティ

目的	プロパティ	例	例の意味
4桁の年	fullYear	2013	西暦2013年
月を表します。0から始まるため、0が1月を表します。	month	11	12月
月の日付を表します。	date	1	1日
曜日を数値で表します。0が日曜日を表します。	day	6	土曜日
時間を0～23の数値で表します。	hours	1：	午後7時
分を0～59の数値で表します。	minutes	3：	30分
秒を0～59の数値で表します。	seconds	0	0秒

SAMPLE CODE

日時を設定する　　DateSampleB.fla

```
//現在日時の取得
var dt:Date = new Date(); ―――――― ①
trace ( dt );
//trace( dt.valueOf() );

//特定の日時をセットする
//Stringでセット( 2015年1月1日 )
var dt1:Date = new Date( "01/01/2015" ); ―――――― ②
trace( dt1 );

//1970/1/1 00:00からの経過ミリ秒で指定
var dt2:Date = new Date( 1300000000000 );
trace( dt2 );

//年、月、日、時、分、秒をそれぞれ指定(2012/11/31 23:59:59)
```

```
var dt3:Date = new Date( 2012, 11, 31, 23, 59, 59 );
trace( dt3 );

//年と月のみ指定(2013年1月)
var dt4:Date = new Date( 2013, 1 );
trace( dt4 );
```


Dateインスタンスに日時を設定するには、コンストラクタにパラメータを渡してすべて行うことができます。①で行っているようにパラメータを渡さない場合、Flash Playerを実行しているOSの現在日付時時刻が設定されます。②のように日まで指定した場合、時刻は0時0分になります。

▶ 実行結果

```
Sun Apr 18 21:08:43 GMT+0900 2010
Thu Jan 1 00:00:00 GMT+0900 2015
Sun Mar 13 16:06:40 GMT+0900 2011
Mon Dec 31 23:59:59 GMT+0900 2012
Fri Feb 1 00:00:00 GMT+0900 2013
Sun Apr 18 2010
Sun Apr 18 2010
Sun Apr 18 2010 09:08:43 PM
Sun Apr 18 21:08:43 GMT+0900 2010
```

2-1-3 日付/時刻の演算を行う

日付の演算を利用して、例えば「ある日付から30日後」というような処理を行うことができます。ActionScriptのDateクラスでは、そのような日付の加算や減算を行うことができます。Dateクラスの日時データはミリ秒（1秒の1/1000）単位で保持されているため、演算にはミリ秒との相互変換が必要になります。Dateクラスで日付/時刻の加算を行うには、次の手順でDateオブジェクトに日時を加算します。Dateクラスで日付/時刻の加算を行う手順は下表のようになります。

Dateクラスで日付/時刻の加算を行う手順

No.	作業内容
1	加算する日数＋時間をミリ秒に変換する。ここで加算する日時の単位により、下表「日時 =>ミリ秒の変換公式」のような変換が必要になります。
2	DateオブジェクトのgetTime()メソッドの戻り値に上記1の結果を加算します。
3	2の結果をもとにして新しいDateオブジェクトを作成する。2-1-1で解説したように、Dateクラスのコンストラクタは1970/1/1からのミリ秒を日付に変換します。

日時 => ミリ秒の変換公式

単位	変換（ミリ秒にするには）	例
秒	1000を掛けます	50秒=> 50×1000ミリ秒
分	1000×60を掛けます	5分=>5×1000×60ミリ秒
時間	1000×60×60を掛けます	3時間 => 3×1000×60×60ミリ秒
日	1000×60×60×24を掛けます	10日 => 10×1000×60×60×24ミリ秒
週	1000×60×60×24×7を掛けます	2週間=> 2×1000×60×60x24×7ミリ秒

※ 1か月と1年は日数が異なるので、公式では固定できません。

Dateクラスの日付/時刻の演算を行うためのメソッド詳細

メソッド	メソッドの詳細
`getTime():Number`	Dateオブジェクトが持つ日時の、1970/1/1 0:00からのミリ秒数(Number)を返します。

SAMPLE CODE

シンプルな日時の加算

DateSampleC.fla

```
//基準日(2011/1/15 11:00)のオブジェクトを作成
var startDate:Date=new Date(2011,1,15,11,0,0,0);

// 1. 加算する日数+時間(30日と18時間)をミリ秒に変換する
var plusDateTime:Number = ——①
     0 * 1000*60*60*24*7 +  // 週
    30 * 1000*60*60*24 +    // 日
    18 * 1000*60*60 +       // 時間
     0 * 1000*60 +          // 分
     0 * 1000;              // 秒

//  2. DateオブジェクトのgetTime()メソッドの戻り値に1. の結果を加算する
var newDateNum:Number=startDate.getTime()+plusDateTime; ——②

//  3. 2の結果をもとにして新しいDateオブジェクトを作成する
var newDate:Date=new Date(newDateNum); ——③
trace( newDate.toString() );
```

表「Dateクラスで日付/時刻の加算を行う手順」をもとに、日数＋時間をDateオブジェクトに追加します。この例では、2011年1月15日11時0分から30日と18時間後の日時を算出します。

①のplusDateTime:Number変数に加算する日数＋時間のミリ秒換算値を代入します。②のnewDateNum:Number変数に新しい日時の1970年からのミリ秒値が代入されます。③でその値から日時を持つDateオブジェクトを作成します。

▶ 実行結果

```
Fri Mar 18 05:00:00 GMT+0900 2011
```

SAMPLE CODE

関数による日時加算処理

DateSampleD.fla

```
//基準日(2011/1/15 11:00)のオブジェクトを作成
var startDate:Date=new Date(2011,1,15,11,0,0,0);
var newDate1:Date = addDays( startDate, 30 ); //30日を加算
var newDate:Date = addHours( newDate1,  18 ); //18時間を加算
trace( newDate.toString() );
```

```
//週を加算する関数
function addWeeks(date:Date, weeks:Number):Date {
  return addDays(date, weeks*7);
}

//日を加算する関数
function addDays(date:Date, days:Number):Date {
  return addHours(date, days*24);
}

//時間を加算する関数
DateSampleC.fla    function addHours(date:Date, hrs:Number):Date {
  return addMinutes(date, hrs*60);
}

//分を加算する関数
function addMinutes(date:Date, mins:Number):Date {
  return addSeconds(date, mins*60);
}

//秒を加算する関数
function addSeconds(date:Date, secs:Number):Date {
  var mSecs:Number=secs*1000;
  var sum:Number=mSecs+date.getTime();
  return new Date(sum);
}
```

手順2の処理（「Dateクラスで日付/時刻の加算を行う手順」のNo.2）を毎回行うのはあまり効率的ではありません。このような処理は関数を作成して行うようにします。ActionScriptリファレンスでも次のサンプルにあるような関数を例示しています。このサンプルでは、上記のサンプルと同じ日時を、関数を使用して加算しています。

▶ 実行結果

```
Fri Mar 18 05:00:00 GMT+0900 2011
```

2-2 タイマーと経過時間

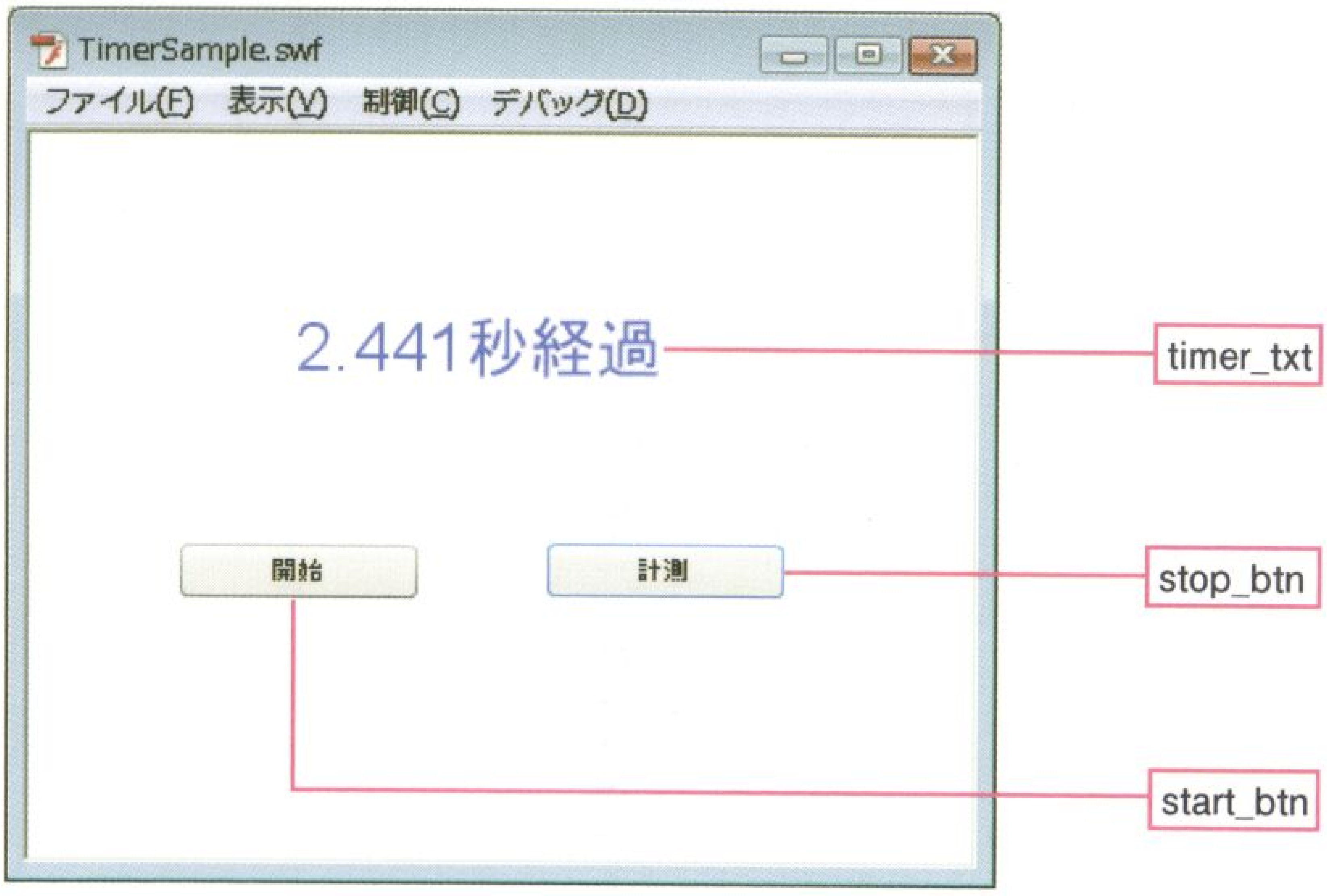
2-2-1 時間を計測する

日単位の時間ではDateクラスを使用しましたが、秒、ミリ秒単位の時間計測についてはタイマー制御関数を利用すると便利です。ActionScript2.0で使用されるタイマー制御関数はActionScript3.0ではflash.utilパッケージに格納されています。その中からgetTimer()関数を解説します。

ActionScriptのタイマー制御関数

関数	関数の詳細
getTimer():int	SWFの再生を開始してからの経過時間(ミリ秒単位)を返します。

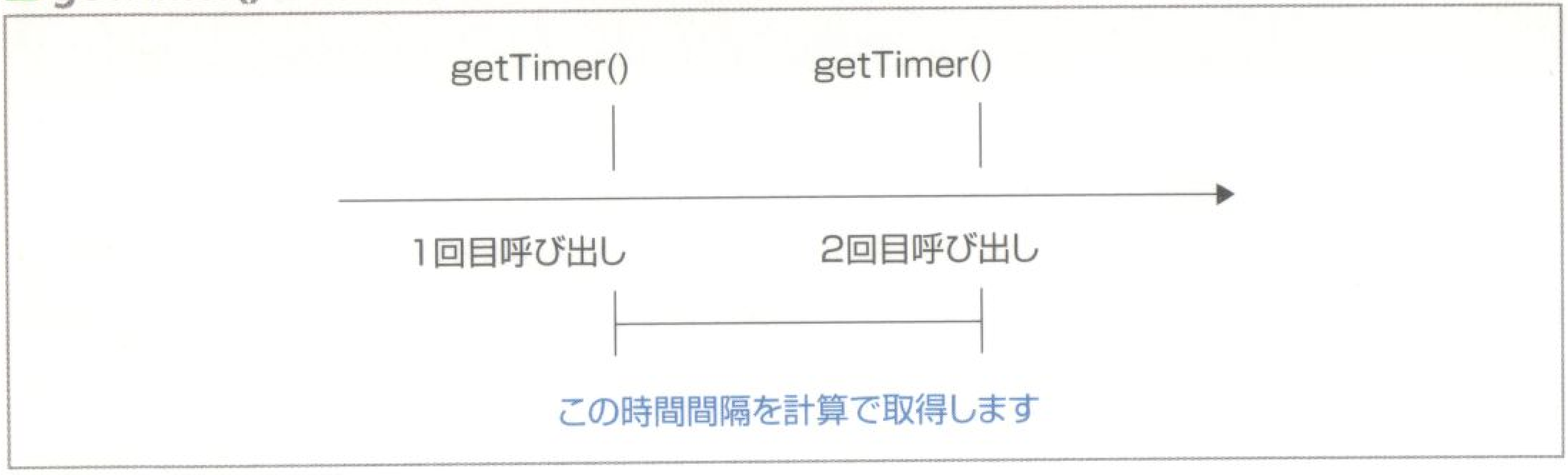
SAMPLE



2回目以降のgetTimer()呼び出しと1回目呼び出し時の時間差を計測して表示するシンプルなサンプルです。start_btnというButton(開始ボタン)を押すことで計測を開始し、stop_btnというButton(計測ボタン)を押すごとに経過時間を計算してtimer_txtというテキストフィールドに秒単位で表示します。

getTimer()関数を2回実行してその差を計算すると、1回目と2回目の間の時間間隔が取得できます。

■ getTimer()を2回実行してその間の経過時間を取得



SAMPLE CODE

getTimer()関数によるシンプル時間計測

TimerSampleA.fla

```
//getTimer関数のインポート
import flash.utils.getTimer;

//スタート時間を保持する変数
var startTime:Number;

//ボタンイベントハンドラの設定
this.start_btn.addEventListener( MouseEvent.CLICK, getStart );
this.stop_btn.addEventListener( MouseEvent.CLICK, getTime );
function getStart( me:MouseEvent ):void {
  this.timer_txt.text = "開始しました";
  //起動後の時間をstartTimeに格納
  startTime = getTimer();
}
function getTime( me:MouseEvent ):void {
  //現在の起動後時間とstartTimeの差から経過時間を算出
  var dif:Number = getTimer() - startTime;
  this.timer_txt.text = ( dif / 1000 ).toString() + "秒経過";
}
```

①startTime変数の宣言
②getTimer()呼び出し1回目
③getTimer()呼び出し2回目

①で宣言したstartTime変数に②の1回目のgetTimer()の結果を格納します。③で行っているように、2回目以降のgetTimer()呼び出し時に差を計算します。

2-2-2 タイマーを使用して繰り返し処理を行う

ActionScriptによるタイマー処理のためにTimerクラスが用意されています。flash.util.Timerクラスは、指定した時間間隔で処理を実行するためのタイマー機能を提供します。イベントの発生を開始するためにstart()メソッドを実行します。このイベントを利用して一定間隔で繰り返し処理を行うことができます。Timerクラスのインスタンスを作成するときに、コンストラクタのパラメータで時間間隔（delay）とtimerイベントの回数（repeatCount）を指定できます。実際のタイマー処理は、Timerオブジェクトのイベントを使用してハンドラ内で行います。Timerクラスには下表のようなイベントが用意されています。

Timerクラスの時間計測用イベント

イベント（下段は文字列定数）	内容
timer TimerEvent.TIMER	指定した時間間隔で繰り返し送出されます。
timerComplete TimerEvent.COMPLETE	指定した回数timerイベントを発生させた後に送出されます。

Timerクラスの計測を制御するためのメソッド詳細

メソッド	メソッドの詳細
Timer(delay:Number, repeatCount:int = 0)	コンストラクタです。パラメータで指定されたタイマー内容を持つオブジェクトを生成します。 **delay:Number** timerイベントを発生させる時間間隔をミリ秒で指定します。 **repeatCount:int** timerイベントを発生させる回数を整数で指定します。指定しない場合、timerイベントは停止させるまで無限に発生します。
start():void	Timerを開始します。 Timerインスタンスの稼働を開始してイベントを発生させるためには、start()メソッドを呼び出す必要があります。停止するにはstop()メソッドを実行します。
stop():void	Timerを終了します。 Timerインスタンスの稼働を停止するにはstop()メソッドを実行します。

SAMPLE CODE

Timerによるシンプルな時間計測

TimerSampleB.fla

```
//1秒で5回イベントを発生するTimerオブジェクトを作成
var timer:Timer=new Timer(1000,5); ―――― ①
var counter:int=0;
```

```
//timerイベントハンドラを設定
timer.addEventListener( TimerEvent.TIMER, timerListener );
//timerCompleteイベントハンドラを設定
timer.addEventListener( TimerEvent.TIMER_COMPLETE, timerComplete );
//Timerを開始
timer.start(); ――――――――――――――――――――――――――――――――――――――――②
//現在時刻を取得して表示
var start:Date = new Date(); ―――――――――――――――――――――――――――――③
trace( "開始時刻:" +  start.getHours() + ":" + start.getMinutes() + " "
+ start.getSeconds() + "." + start.getMilliseconds() );
function timerListener( te:TimerEvent ):void {
  trace( "timerイベント:" + counter );
  counter++;
}
function timerComplete( te:TimerEvent ):void { ――――――――――――――――④
  var end:Date = new Date(); ――――――――――――――――――――――――――――⑤
  trace( "終了時刻:" +  end.getHours() + ":" + end.getMinutes() + " " +
  end.getSeconds() + "." + end.getMilliseconds() );
  timer.removeEventListener( TimerEvent.TIMER, timerListener );
}
```

Timerクラスのコンストラクタで時間間隔と繰り返し回数を指定するシンプルなサンプルです。timerCompleteイベントハンドラでtimerイベントハンドラの登録を解除します（④）。この解除処理はサンプルでは必ずしも必要ではありませんが、実際のアプリケーションではリソース節約やバグ防止のためにも、行うようにしましょう。

①のTimerクラスコンストラクタで1000ミリ秒（1秒）、5回繰り返しの設定を行っています。②でstart()メソッドを呼び出してTimerを開始しています。この後1秒ごとにtimerListener関数が呼び出されます。③と⑤のDateオブジェクトは、開始/終了時刻を取得するためです。

▶ 実行結果

```
開始時刻:10:35 50.32
timerイベント:0
timerイベント:1
timerイベント:2
timerイベント:3
timerイベント:4
終了時刻:10:35 55.36
```

PART 3



数値と数学処理

3-1-1 数値データを使う

ActionScript3.0では数値データ型はすべてクラスで提供されます。そのため、数値データはオブジェクトです。しかし数値データは単一の値を持つシンプルなデータであり、数値演算の対象にもなるため、Arrayなどの複合型オブジェクトよりも処理効率を上げるための仕組みが取り入れられています。この特殊なオブジェクトは「プリミティブデータ型」と呼ばれ、Boolean、int、Number、String、uintが該当します。このうちint、Number、uintが数値データ型です。

ActionScript3.0の数値データ型詳細

データ型	詳細
int	プラス、マイナスの整数値を格納できます。ActionScriptでは、32ビットを使用するため、-2^{31} ～ $2^{31}-1$の範囲が有効です。この範囲を超える数値か、実数を扱う場合、Number型を使用します。 値の範囲はint.MIN_VALUE、int.MAX_VALUEというパブリック定数でも定義されています。 ※-2^{31} = −2,147,483,648　$2^{31}-1$ = 2,147,483,647
Number	プラス、マイナスの整数と小数つきの実数値を格納できます。64ビットが使用されます。値の範囲はNumber.MAX_VALUE 、Number.MIN_VALUEというパブリック定数で定義されています。ActionScriptのNumber型では、例えば1/3という分数を処理する場合などでは近似値による演算となります。int、uintの2倍のビット数が使用されるため、整数のみに使用する変数についてはNumber型の使用は避けたほうが実行効率が上がります。
uint	プラスの整数値を格納できます。int型と同様に32ビットします。int型と異なりマイナス値を扱えない代わりに、扱えるプラス数値の範囲が約2倍になります。0 ～ $2^{32}-1$の範囲に使用できます。uintのuは、unsigned（符号なし）を意味します。値の範囲はuint.MIN_VALUE、uint.MAX_VALUEというパブリック定数でも定義されています。 ※$2^{32}-1$ = 4,294,967,295

SAMPLE CODE

数値データ型変数の宣言

```
var intData:int = initialValue;  // newとコンストラクタを省略
var intData:int = new int( initialValue ); //newとコンストラクタを使用
```

ActionScrip3.0の数値データ型はクラスで提供されますが、変数宣言とインスタンスの生成にnew演算子とコンストラクタは使用しても、しなくてもかまいません。下記の2つのint型変数の宣言と初期化は全く同じです。initValueは整数値を表します。

3-1-2 数値を指定の書式に変換する

3-1-1「数値データを使う」で解説したように、ActionScript3.0では数値データ型はすべてクラスで提供されるため、数値データ自体のメソッドを使用できます。int、Number、uint各クラスに共通するメソッドが提供されており、それらを使用して書式の変換を行うことができます。メソッドと用途を下記の表にまとめます。

■ ActionScript3.0数値データクラスの主な共通メソッド

目的の書式	メソッド	メソッドの詳細
指数表現	toExponential(fractionDigits:uint):String	数値データの指数表現形式を作成します。指数表現とは、9.99e+3 のような表現で、e+3は「$\times10^3$」を表します。つまり9990と同値です。 9.99e−3のようにマイナスがつく場合は「$\times10^{-3}$」を表し、この場合0.00999と同値になります。 パラメータのfractionDigitsは小数点以下の桁数を表し、0以上20以下まで指定できます。
固定小数点表現	toFixed(fractionDigits:uint):String	小数点以下の桁数を指定の長さにした形式を作成します。実際の桁数が指定した長さより長い場合、切り捨てられ、短い場合は0が挿入されます。 パラメータのfractionDigitsで小数点以下の桁数を指定します。
桁数指定	toPrecision(precision:uint):String	指定した桁数で数値を表記します。指数表現が必要な場合、作成されます。 パラメータのprecisionで全体の桁数を指定します。
2進～36進数表記	toString(radix:Number = 10):String	数値のストリング表現形式を作成します。 パラメータのradixで基数（進数）を指定できます。デフォルト値が10なので、指定しない場合10進数の表記となります。

SAMPLE CODE

数値の基本的な書式変換 NumbersSample.fla

```
var i:int = -101;
var n:Number = 123.4567;
var ui:uint = 1001;
//toString()
trace( "-toString ( 10 ) :10進表記 ---- " );  ─┐
trace( "i:int = " + i.toString( 10 ) );         │
trace( "n:Number = " + n.toString( 10 ) );      ├─ ①
trace( "ui:uint = " + ui.toString( 10 ) );     ─┘
//toExponential ()
```

```
trace( "-toExponential  ( 2 ) : 小数点以下2桁の指数表現----   " );
trace( i.toExponential( 2 ) );
trace( n.toExponential( 2 ) );
trace( ui.toExponential( 2 ) );
//toFixed()
trace( "-toFixed  ( 2 ) : 小数点以下2桁の固定小数点表現----   " );
trace( i.toFixed( 2 ) );
trace( n.toFixed( 2 ) );
trace( ui.toFixed( 2 ) );
//toPrecision()
trace( "-toPrecision  ( 2 ) : 2桁の桁数指定  ---- " );
trace( i.toPrecision( 2 ) );
trace( n.toPrecision( 2 ) );
trace( ui.toPrecision( 2 ) );
```

(②: toExponential の4行、③: toFixed の4行、④: toPrecision の4行)

int、Number、uintの各オブジェクトについて、共通のメソッドによる書式表現を確認するサンプルです。

①で、toString(10)により10進数に変換した値を出力しています。②がtoExponential()による指数表現、③がtoFixed()による固定小数点表現、④がtoPrecision()による全体桁数指定の表現です。

▶ 実行結果

```
-toString  ( 10 ) :10進表記 ----
i:int = -101
n:Number = 123.4567
ui:uint = 1001
-toExponential  ( 2 ) : 小数点以下2桁の指数表現----
-1.01e+2
1.23e+2
1.00e+3
-toFixed  ( 2 ) : 小数点以下2桁の固定小数点表現----
-101.00
123.46
1001.00
-toPrecision  ( 2 ) : 2桁の桁数指定  ----
-1.0e+2
1.2e+2
1.0e+3
```

3-1-3 数学的な演算を行う

ActionScript3.0でsin()などの数学的な関数やπのような定数を使う場合、Mathクラスを使用します。このクラスのメンバ(プロパティとメソッド)はすべてstatic(静的)です。staticであるということは、new Math()のようにインスタンスを生成しなくてもMath.sin()のようにクラス名でアクセスできることを意味します。必要なときに関数や定数を手軽に利用できるわけです。Mathクラスのメソッドや定数の呼び出しは、下記のように行います。

Mathクラスのプロパティとメソッドの呼び出し構文

```
Math.method( parameter );  // Mathクラスのメソッド呼び出し
Math.constant //Mathクラスのプロパティ(定数)へのアクセス
```

Mathクラスの主な定数プロパティ

目的	定数プロパティ
自然対数の底	E:Number
円周率	PI:Number

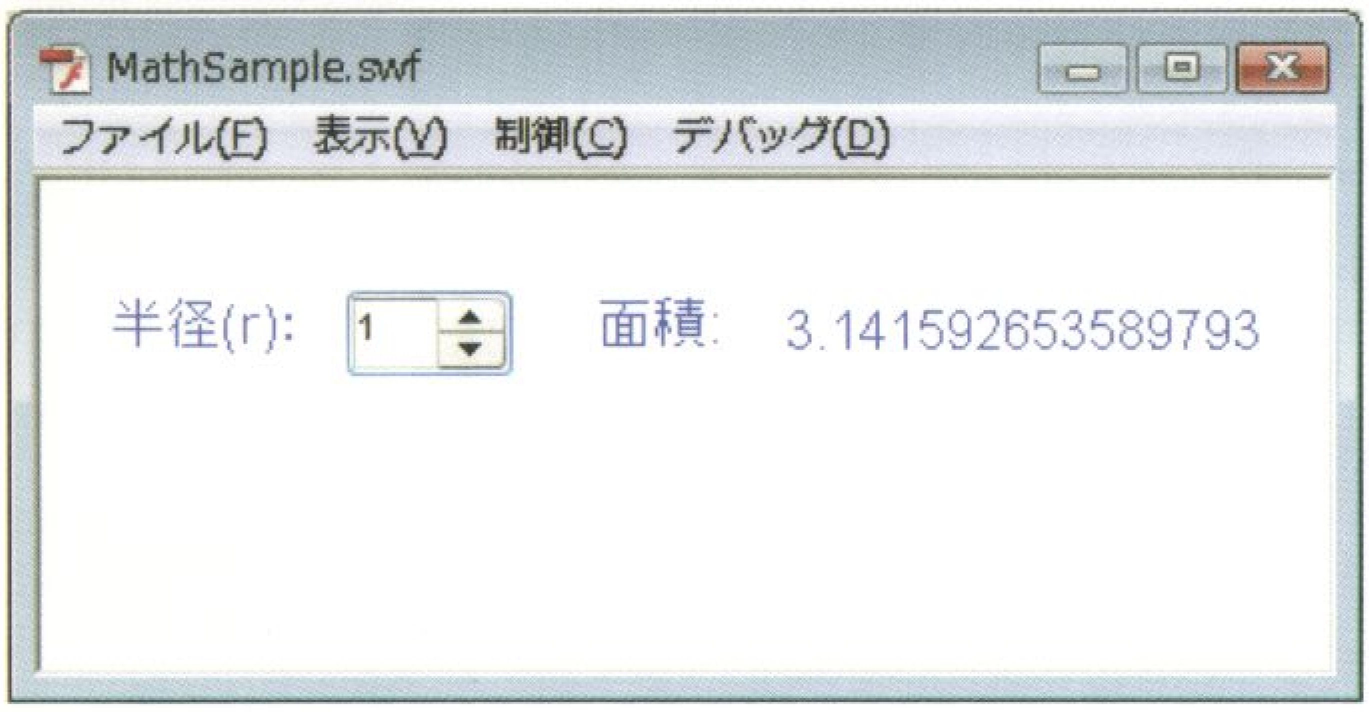
Mathクラスの一般的な数学処理のためのメソッド

目的	メソッド	メソッドの戻り値	サンプル式	サンプル式の結果
絶対値	`abs( val:Number ):Number`	valの絶対値	abs(-10)	10
小数点切り上げ	`ceil( val:Number ):Number`	valの小数点を切り上げた値	ceil(9.5)	10
小数点切り下げ	`floor( val:Number ):Number`	valの小数点を切り下げた値	floor(9.5)	9
最大値	`max( v1:Number, v2:Number,..rest ):Number`	パラメータの中で最大の値	max(1.2, 1.0, 1.3)	1.3
最小値	`min( v1:Number, v2:Number,..rest ):Number`	パラメータの中で最小の値	min(1.2, 1.0, 1.3)	1.0

Mathクラスの一般的な数学処理のためのメソッド（続き）

目的	メソッド	メソッドの戻り値	サンプル式	サンプル式の結果
累乗（べき）	pow(v1:Number, v2:Number):Number	v1をv2乗した値	pow(3, 2)	9
ランダムな数	random() :Number	0以上1未満のランダムな数値	random()	乱数
四捨五入	round(val:Number):Number	valを四捨五入した整数値	val(3.45)	3
平方根	sqrt(val:Number):Number	valの平方根	val(81)	9

SAMPLE



fl.controls.NumericStepperコンポーネントを使用するサンプルです。NumericStepperで選択する半径の値をもとに、pow()関数と円周率を表すMath.PI定数を使用して面積を計算します。

SAMPLE CODE

半径の値をもとに面積を計算する MathSample.fla

```
area_txt.text = "0";
//NumericStepperが変化したときgetArea関数を呼び出します
r_ns.addEventListener( Event.CHANGE, getArea );
function getArea( event:Event ):void {
  //NumericStepperの値とMath.PI 定数により面積を計算します
  area_txt.text = ( Math.pow( r_ns.value, 2 ) * Math.PI ).toString();
}
```

3-1-4 ランダムな数値を生成する

Flashアプリケーションでおみくじや抽選を行う場合などに必要となるのがランダムな数値の生成です。ランダムな数値はMathクラスのrandom()メソッドにより生成できます。Mathクラスの数学関数と定数はすべてstatic属性で提供されるため、インスタンスを生成することなくそれらを使うことができます。

Mathクラスのランダムな数値を発生するためのメソッド

メソッド	メソッドの詳細
random():Number	0以上1未満のランダムな数値を返します。このメソッドが生成する数値は計算式により算出されているため、人がサイコロを振るときのように完全なランダム値ではなく「疑似乱数」と呼ばれます。

SAMPLE CODE

0以上100以下のランダムな数値発生させる

MathRandomSample.fla

```
var lot:Number;
trace("乱数を使用したくじ引きを開始します");
lot = Math.random() * 100; ――――― ①
lot = Math.ceil( lot ); ――――― ②
trace("結果を発表します。");
if ( lot >= 80 ) {
  trace("当たりです！　点数は" + lot + "点でした。");
} else {
  trace("はずれです。 点数は" + lot + "点でした。");
}
```

random()メソッドは1未満の数を返すので、使用する目的に応じて整数化するなどの処理を行います。個のサンプルでは、0以上100以下の整数を算出して80以上では当たり、80未満でははずれとしています。

①の計算でlot変数には0以上100未満の値が格納されます。②のceil()関数で小数切り上げを行い整数化しています。

▶ 実行結果

```
乱数を使用したくじ引きを開始します
結果を発表します。
当たりです！　点数は87点でした。
```

3-1-5 四捨五入や小数の切り捨て、切り上げを行う

PART
1
2
3
4
5
6
7
8
9

Flashアプリケーションでロードされたswfファイルのパーセンテージを算出するときなど、小数点以下の数値を切り捨てたり四捨五入する、というようないわゆる「数値を丸める」処理の必要性が出てきます。数値の丸め処理はMathクラスに用意されたメソッドで行うことができます。

■ Mathクラスの四捨五入、小数切り捨て/切り上げを行うためのメソッド詳細

メソッド	メソッドの詳細
ceil(val:Number):Number	パラメータで与えられる数値(val)を、その数値以上のもっとも近い整数に切り上げた値を返します。
floor(val:Number):Number	パラメータで与えられる数値(val)を、その数値以下のもっとも近い整数に切り下げた値を返します。
round(val:Number):Number	パラメータで与えられる数値(val)を、もっとも近い整数に四捨五入した値を返します。2つの整数値のちょうど中間にある場合、大きいほうの整数に切り上げられます。

SAMPLE CODE

数値を丸める基本処理 MathRoundSample.fla

```
var num1:Number = 1.499;
var num2:Number = 1.5; ──①
var num3:Number = 1.501;
trace( "num1="+num1+", num2="+num2+",num3="+num3 );
trace( "ceilメソッドによる切り上げ: " + "num1="+Math.ceil(num1)+",
        num2="+Math.ceil(num2)+",num3="+Math.ceil(num3) );
trace( "floorメソッドによる切り下げ: " + "num1="+Math.floor(num1)+",
        num2="+Math.floor(num2)+",num3="+Math.floor(num3) );
trace( "roundメソッドによる四捨五入: " + "num1="+Math.round(num1)+",
        num2="+Math.round(num2)+",num3="+Math.round(num3) );
trace( "マイナス値の処理です" );
num1 = -1.499;
num2 = -1.5;
num3 = -1.501;
trace( "num1=" + num1 + ", num2=" + num2+",num3=" + num3 );
trace( "ceilメソッドによる切り上げ: " + "num1="+Math.ceil(num1) + ",
        num2="+Math.ceil(num2) + ",num3=" + Math.ceil(num3) );
trace( "floorメソッドによる切り下げ: " + "num1="+Math.floor(num1)+",
```

```
num2="+Math.floor(num2)+",num3="+Math.floor(num3) );
trace( "roundメソッドによる四捨五入: " + "num1="+Math.round(num1)+",
num2="+Math.round(num2)+",num3="+Math.round(num3) );
```

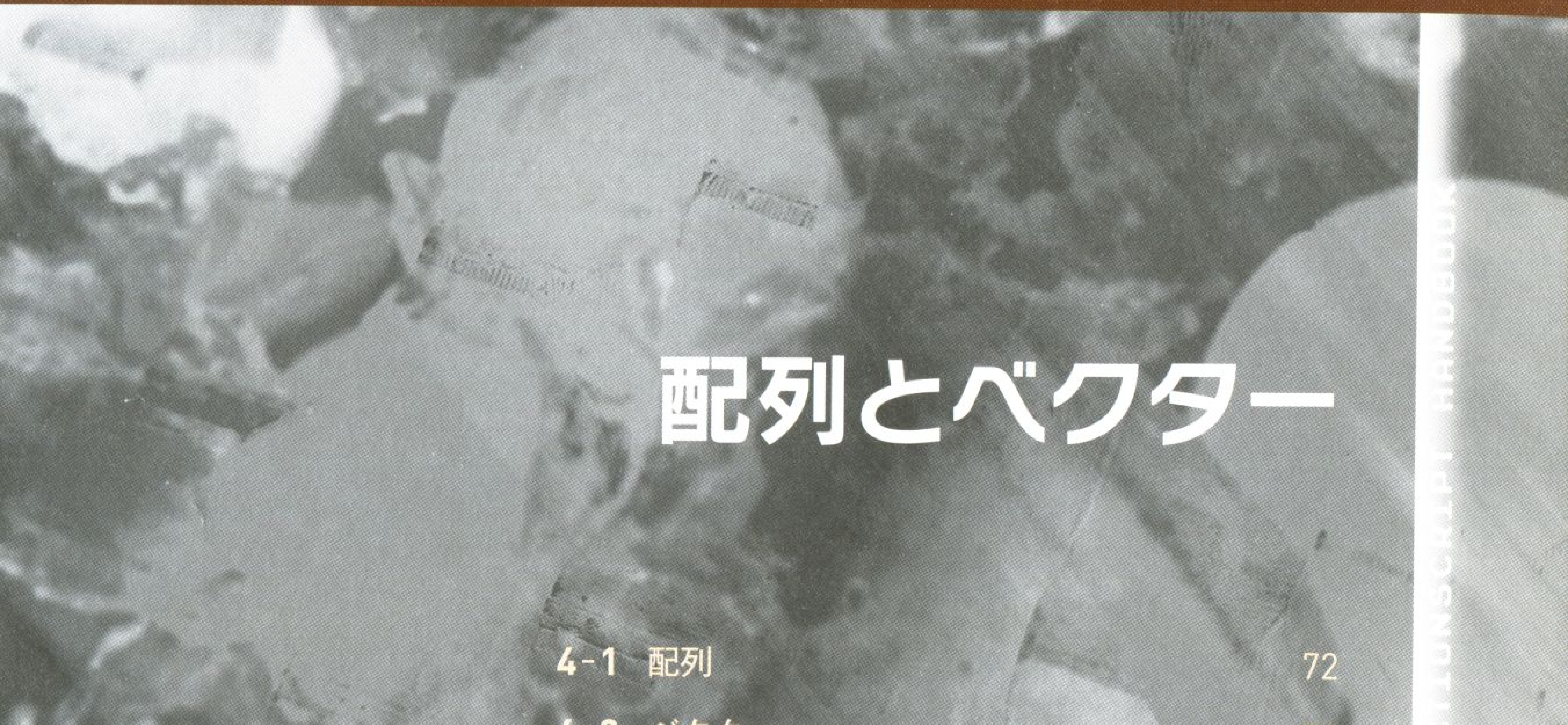
四捨五入や切り捨て、切り上げ処理で勘違いしやすいのがマイナス値の処理です。サンプルではマイナス値の処理も含めています。

①のnum2に設定した1.5という数値は1と2のちょうど中間になるため、roundメソッドの仕様通り、切り上げが行われています。その際、マイナス値(−1.5)の場合結果は−1になります。

▶ 実行結果

```
num1=1.499, num2=1.5,num3=1.501
ceilメソッドによる切り上げ: num1=2, num2=2,num3=2
floorメソッドによる切り下げ: num1=1, num2=1,num3=1
roundメソッドによる四捨五入: num1=1, num2=2,num3=2
マイナス値の処理です
num1=-1.499, num2=-1.5,num3=-1.501
ceilメソッドによる切り上げ: num1=-1, num2=-1,num3=-1
floorメソッドによる切り下げ: num1=-2, num2=-2,num3=-2
roundメソッドによる四捨五入: num1=-1, num2=-1,num3=-2
```

PART 4



配列とベクター

4-1-1 インデックス配列を使う

ActionScriptで配列を扱う場合、下表「ActionScript3.0配列の種類と使用するクラス」にあるように、インデックス配列と結合配列があり、使用するクラスも違うものになります。インデックス配列は0から始まる整数インデックスで要素を管理し、結合配列は数値ではなく文字列やオブジェクトをキーとします。表にあるように、インデックス配列はArrayクラスを使用して操作します。ここではArrayクラスによるインデックス配列の操作を解説します。

ActionScript3.0配列の種類と使用するクラス

配列の種類	内容	クラス
インデックス配列	先頭の要素が0で始まる整数のインデックスで要素を管理する一般的な配列です。	Array
結合配列	ストリング値をキー（要素を識別する値）とする結合配列を作成できます。	Object
	任意のデータ型をキーとする結合配列を作成できます。	Dictionary

インデックス配列のArrayオブジェクトを作成するには、下表のような方法があります。

インデックス配列のArrayオブジェクトを作成する方法

No.	方法	サンプル
1	配列リテラルによる初期化	var ar:Array = ["要素0", "要素1", "要素2"]; trace(ar); // 要素0, 要素1, 要素2
2	コンストラクタによる初期化	var ar:Array = new Array("要素0", "要素1", "要素2");; trace(ar); // 要素0, 要素1, 要素2
3	コンストラクタで要素のみを確保する	var ar:Array = new Array(3);; //要素数=3の配列を作成します。 ar[1] = "要素1"; //2番目の要素に値をセット trace(ar); //,要素1
4	メソッドで要素を追加	var ar:Array = new Array(); //要素数=0 の配列を作成します。 ar.push("要素1"); trace(ar); // 要素1

Arrayクラスの主なプロパティ

目的	プロパティ名	プロパティの概要
配列の要素数を得る	`length`: unit	配列の要素数を示す整数値です。要素がない配列の場合、0になります。

Arrayクラスのインデックス処理のための主なメソッド

目的	メソッド名	メソッドの概要
フィルターをかける	filter	Boolean値を返す関数を作成して、その関数がtrueを返す値を要素として持つ新しい配列を返します。
各要素に対して関数を実行する	forEach	関数を作成して配列の各要素に対して実行できます。
最後の要素を取り出して削除する	pop	配列の最後の要素を削除します。メソッドの戻り値でその要素が返されます。配列の要素数(length)は減少します。
最後に要素を追加する	push	要素を配列の末尾に追加します。配列の要素数(length)は増加します。
要素の並びを反転させる	reverse	先頭から末尾の並び順を反転させます。
先頭の要素を取り出して削除する	shift	配列の先頭の要素を削除します。メソッドの戻り値でその要素が返されます。配列の要素数(length)は減少します。
ソートする	sort	要素をUnicode値により昇順ソートします。アルファベットでは大文字-小文字の順になります。比較を行うための独自の関数を作成することもできます。
要素を追加/削除する	splice	先頭のインデックスと要素数を指定して配列の要素を追加/削除できます。
先頭に要素を追加する	unshift	要素を配列の先頭に追加します。配列の要素数(length)は増加します。
要素をすべて文字列にする	toString	配列内の要素を「,」で区切られた文字列で返します。

1
1

SAMPLE CODE

基本的なインデックス配列の処理

ArraySampleA.fla

```
var ar:Array = [ "要素0", "要素1", "要素2" ];   //オブジェクトの作成
trace( ar );  // 要素0, 要素1, 要素2
trace( ar.length );  // 3
//末尾に要素を追加
ar.push( "要素3" );
trace( ar );  // 要素0, 要素1, 要素2, 要素3
trace( ar.length );  // 4
//先頭の要素を削除
var s:String = ar.shift();
trace( ar );  //  要素1, 要素2, 要素3
trace( ar.length );  // 3
//並び順を逆転
ar.reverse();
trace( ar );  //  要素3, 要素2, 要素1
```

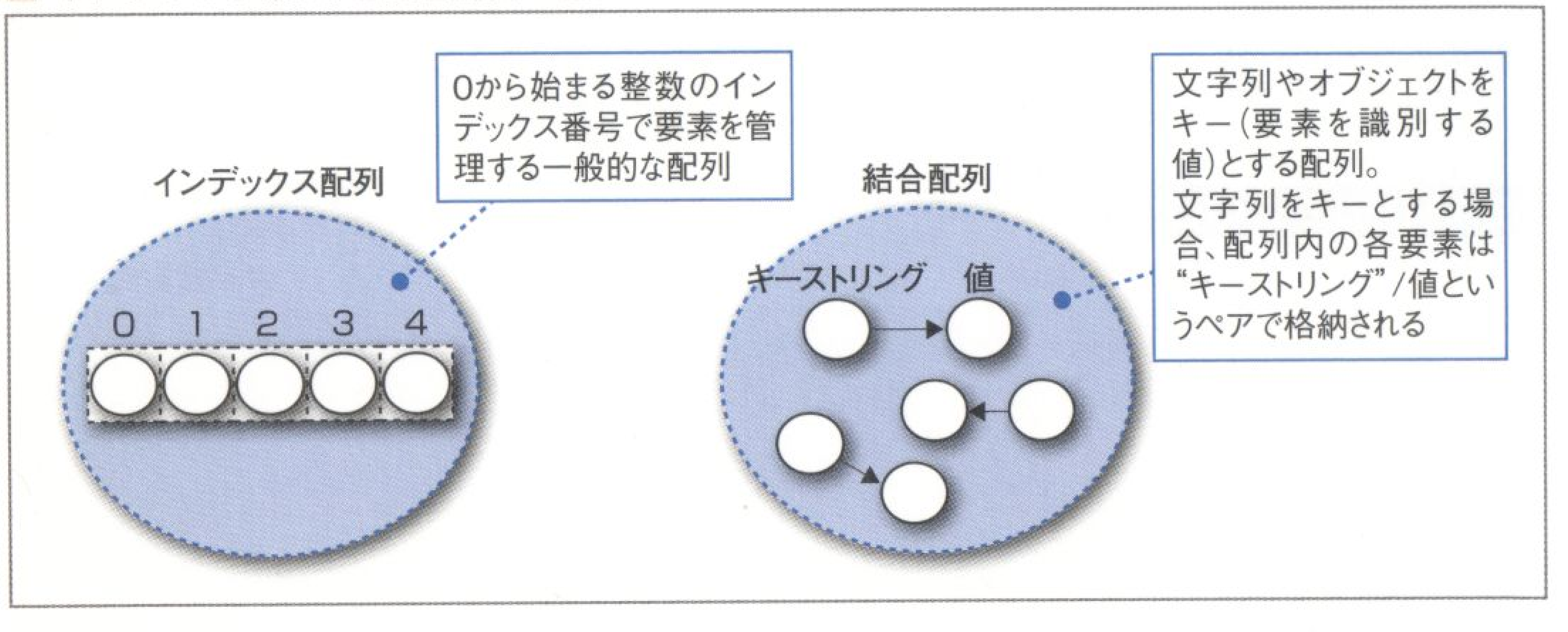
```
//ソート
ar.sort();
trace( ar );  //  要素1, 要素2, 要素3
```


lengthプロパティと基本的なメソッドを使用しています。trace()関数に配列オブジェクトを渡すことで、toString()メソッドが呼び出されてすべての要素を出力できます。

▶ 実行結果

```
要素0,要素1,要素2
3
要素0,要素1,要素2,要素3
4
要素1,要素2,要素3
3
要素3,要素2,要素1
要素1,要素2,要素3
```

■ インデックス配列と結合配列の違い



4-1-2 結合配列を使う

整数値で配列の要素を識別するインデックス配列だけではなく、文字列やオブジェクトをキーとする結合配列を作成できます。ここでは文字列をキーとする結合配列の作成方法と操作を解説します。文字列をキーとする場合、配列内の各要素は"キーストリング" / 値というペアで格納されます。この配列を作成するのにもっとも適切なのがObjectクラスです。つまり、結合配列は通常Objectクラスのインスタンスです。これはObjectクラスがもっともシンプルなdynamicクラスであるためです。

結合配列を作成する方法

No.	方法	サンプル
1	オブジェクトリテラルによる初期化	var myCar:Object = { type:"sports", price:400 };
2	動的に要素を追加	var myCar:Object = new Object(); myCar.type = "sports"; myCar["price"] = 400;

文字列をキーとする結合配列を処理するためのActionScript演算子

演算子	説明	例
[ ] 括弧演算子	[ ] 内にキー値を指定することにより、結合配列の要素にアクセスできます。	obj["key"]=値; trace(obj["key"]);
ドット演算子	. を使用してキー指定することにより、結合配列の要素にアクセスできます。	obj.key = 値 ; trace(obj.key);

SAMPLE CODE

基本的な結合配列の処理

ArraySampleB.fla

```
//オブジェクトリテラルによる結合配列の生成
var myCar:Object = { type:"sports", price:400 };
trace( myCar["type"], myCar["price"] );  // sports 400 ——①[ ]演算子による要素へのアクセス
//動的プロパティとしてキーと要素を追加
var yourCar:Object = new Object();
yourCar.type = "4wd"; ——②動的プロパティとして要素を追加
yourCar["price"] = 450; ——③[ ]演算子による要素へのアクセス
trace( yourCar.type, yourCar.price );  // 4wd 450 ——④ドット演算子による要素の取り出し
```

4-1-3 配列にフィルターをかける

PART
1
2
3
4
5
6
7
8
9

配列内の要素1つ1つに対して関数を実行し、条件に合う要素のみを抽出して新しい配列を作成できます。このような処理はフィルターと呼ばれ、Arrayクラスのfilter()メソッドが用意されています。このメソッドに自作の関数を指定します。この関数には、下記の書式のように各要素の値が渡され、関数がtrueを返す要素のみが新しい配列としてfilter()メソッドの戻り値に含まれます。元の配列は変化しません。

Array クラスのフィルターメソッド詳細

メソッド	メソッドの詳細
`filter( callback:Function, thisObject:* = null ):Array`	**callback**：配列内の各要素に対して実行される関数を指定します。この関数がtrueを返す要素が新しい配列に格納されてメソッドの戻り値になります。 **thisObject**：関数内でthisを使用したときに参照されるオブジェクトを指定できます。デフォルト値=nullなので、指定する必要のない場合は使用しません。

Array クラスのfilter()メソッドに指定する関数の書式

関数	関数の詳細
`function callback( item:*, index:int, array:Array ):Boolean;`	**item**：配列内の要素です。この要素の値に対して関数内で演算や判定を行い、trueを返す要素が抽出されます。 **index**：itemの、配列内でのインデックスです。 **array**：filter()メソッドを実行するArrayオブジェクトです。

SAMPLE CODE

Array クラスのfilter() メソッドを使う

ArraySampleC.fla

```
//インデックス配列オブジェクトの作成
var students:Array = new Array();
students.push( { name:"山田一郎", score: 88 } );
students.push( { name:"鈴木二郎", score: 78 } );
students.push( { name:"須藤俊一", score: 25 } );   ①
students.push( { name:"宮田三郎", score: 44 } );
students.push( { name:"三輪彰夫", score: 63 } );

//filter()メソッドの実行と出力
var goukaku:Array = students.filter( scoreFilter );   ②
```

```
for ( var i:int = 0 ; i < goukaku.length ; i++ ) {
  trace( goukaku[ i ].name + "¥t点数:" + goukaku[ i ].score );
}

//filter関数
function scoreFilter( student:*, index:int, ary:Array ):Boolean {  ③
  if ( student.score >= 60 ) return true;  //scoreが60以上の場合、結果に含
  める  ④
  else return false;
}
```


①ではインデックス配列studentsに要素を追加しています。要素の値の記述は次のようになっており、これにより各要素はname、およびscoreというプロパティを持つObject型の値を持つようになります。

```
{ name:"山田一郎", score:88 }
```

このようにObjectクラスのインスタンスは、

{ プロパティ 1:値1, プロパティ 2:値2 ...}

という書式で生成できます。

②でfilter()メソッドを実行してscoreFilterという関数名を指定しています。

③がscoreFilter関数で、student:* というパラメータに配列の各要素が渡されて繰り返し実行されます。④で要素内のscoreプロパティの値が80以上の場合にtrueを返しています。

▶ 実行結果

```
山田一郎 点数:88
鈴木二郎 点数:78
三輪彰夫 点数:63
```

4-1-4 Dictionaryによる結合配列を使う

ActionScript3.0では、Dictionaryクラスを使用して、オブジェクトをキーとする結合配列を作成できます。オブジェクトをキーとする場合、Dictionaryインスタンスが結合配列オブジェクトとなり、配列内の各要素は、オブジェクト/値というペアで格納されます。Dictionaryによる結合配列を作成する場合、下記のように、[]（演算子）を使用して結合配列の要素にアクセスできます。

オブジェクトをキーとする結合配列を処理するためのActionScript演算子

演算子	説明	例
[ ] 括弧演算子	[ ] 内にキー値を指定することにより、結合配列の要素にアクセスできます。	obj[keyobj]=値; trace(obj[keyobj]);

SAMPLE CODE

Dictionaryによる結合配列を作成する　　ArraySampleD.fla

```
import flash.utils.Dictionary;
var da:Dictionary = new Dictionary();

var key1:Bitmap = new Bitmap();
var key2:Camera = new Camera();        ①
var key3:Date = new Date();

var val1:Array =  [ "10", "20" ];
var val2:String = "ABC";
var val3:Object = new Object();

da[key1] = val1;
da[key2] = val2;        ②
da[key3] = val3;

trace( da[key1]  );
trace( da[key2]  );
trace( da[key3]  );
```

①で作成しているオブジェクトが、Dictionary配列のキーとなるものです。②でそれらのキーを使用して、結合配列に値を格納しています。

4-2 ベクター

4-2-1 ベクターを使う

ActionScriptの配列にはデータ型の異なるオブジェクトを格納できますが、Vectorクラスを使用するベクターと呼ばれる配列は、同じデータ型で構成します。その目的はデータ型の統一による処理の効率化と整合性の確保です。要素がみな同じデータ型であるため、アクセスがArrayよりも高速にできることがその理由です。また、Vectorオブジェクトを作成するときにデータ型を決定するため、そのデータ型以外の要素は格納できなくなります。指定したデータ型は「ベース型」と呼ばれます。Vectorクラスのメソッドとプロパティは Arrayクラスと類似しています。Vectorクラスのオブジェクトを作成する場合、下表の2つの方法があります。

Vectorクラスのオブジェクトを作成する方法

No.	方法	サンプル
1	Vector()グローバル関数でArrayを変換して初期化	var vt:Vector.<T>=Vector.<T>(arrayT); // Tはベース型を表すActionScriptデータ型。arrayTは要素のデータ型すべてTの配列
2	Vectorクラスのメソッドで要素を追加	var vt:Vector.<T>; // Tはベース型を表すActionScriptデータ型 v = new Vector.<T>(); //インスタンスの生成 v.push(item:T); //pushメソッドで要素を追加

ベクターを処理するためのActionScript演算子

演算子	説明	例
[ ] 括弧演算子	[ ] 内にインデックス値を指定することにより、ベクターの要素にアクセスできます。	vtr[0]=値; trace(vtr[0]);

Vectorクラスの主なプロパティ

目的	プロパティ名	プロパティの概要
要素数が固定的なベクターを作る	`fixed:` `Boolean`	trueの場合、Vectorオブジェクトの要素数を変更できません。つまり、push()、pop()などのメソッドが発行できなくなります。
ベクター内の要素数を得る	`length:` `uint`	ベクターの要素数を示す整数値です。要素がないベクターの場合、0になります。

Vectorクラスの主なメソッド

目的	メソッド名	メソッドの概要
フィルターをかける	`filter`	Boolean値を返す関数を作成して、その関数がtrueを返す値を要素として持つ新しいベクターを返します。
各要素に対して関数を実行する	`forEach`	関数を作成してベクターの各要素に対して実行できます。

Vectorクラスの主なメソッド（続き）

目的	メソッド名	メソッドの概要
最後の要素を取り出して削除する	pop	ベクターの最後の要素を削除します。メソッドの戻り値でその要素が返されます。ベクターの要素数（length）は減少します。
最後に要素を追加する	push	要素をベクターの末尾に追加します。ベクターの要素数（length）は増加します。
要素の並びを反転させる	reverse	先頭から末尾の並び順を反転させます。
先頭の要素を取り出して削除する	shift	ベクターの先頭の要素を削除します。メソッドの戻り値でその要素が返されます。ベクターの要素数（length）は減少します。
ソートする	sort	比較を行う関数を作成し、その結果に基づくソートを実施します。
要素を追加/削除する	splice	先頭のインデックスと要素数を指定してベクターの要素を追加/削除できます。
先頭に要素を追加する	unshift	要素をベクターの先頭に追加します。ベクターの要素数（length）は増加します。
要素をすべて文字列にする	toString	ベクター内の要素を「,」で区切られた文字列で返します。

SAMPLE CODE

基本的なベクターの処理 VectorSampleA.fla

```
var vt:Vector.<String> = Vector.<String>(["要素0", "要素1", "要素2"]);//
オブジェクトの作成
trace( vt );  // 要素0, 要素1, 要素2
trace( vt.length );  // 3
//末尾に要素を追加
vt.push( "要素3" );
trace( vt );  // 要素0, 要素1, 要素2, 要素3
trace( vt.length );  // 4
//先頭の要素を削除
var s:String = vt.shift();
trace( vt );  //  要素1, 要素2, 要素3
trace( vt.length );  // 3
//並び順を逆転
vt.reverse();
trace( vt );  //  要素3, 要素2, 要素1
```

lengthプロパティと基本的なメソッドを使用しています。trace()関数に配列オブジェクトを渡すことで、toString()メソッドが呼び出されてすべての要素を出力できます。

4-2-2 ベクターの要素をフィルターする

ベクター内の各要素に対して関数を実行し、条件に合う要素のみを抽出して新しいベクターを作成できます。このようなフィルター処理を行うために、Vectorクラスのfilter()メソッドが用意されています。このメソッドに設定する関数には各要素の値が渡され、関数がtrueを返す要素のみが新しいベクターとしてfilter()メソッドの戻り値に含まれます。元のベクターは変化しません。

Vectorクラスのフィルターメソッド詳細

メソッド	メソッドの詳細
`filter( callback:Function, thisObject:* = null ):Vector.<T>`	**callback**：ベクター内の各要素に対して実行される関数を指定します。この関数がtrueを返す要素が新しいベクターに格納されてメソッドの戻り値になります。 **thisObject**：関数内でthisを使用したときに参照されるオブジェクトを指定できます。

Vectorクラスのfilter()メソッドに指定する関数の書式

関数	関数の詳細
`function callback( item:T, index:int, vector:Vector.<T> ):Boolean;`	**item**：ベクター内の要素です。この要素の値に対して関数内で演算や判定を行い、trueを返す要素が抽出されます。 **index**：itemの、ベクター内でのインデックスです。 **vector**：filter()メソッドを実行するVectorオブジェクトです。

SAMPLE CODE

Vectorクラスのfilter()メソッドを使う

VectorSampleB.fla

```
//ベクターオブジェクトの作成
var students:Vector.<Object> = new Vector.<Object>();
students.push( { name:"山田一郎", score: 88 } );
students.push( { name:"鈴木二郎", score: 78 } );
students.push( { name:"須藤俊一", score: 25 } );
students.push( { name:"宮田三郎", score: 44 } );
students.push( { name:"三輪彰夫", score: 63 } );

//filter()メソッドの実行と出力
var goukaku:Vector.<Object> = students.filter( scoreFilter );
for ( var i:int = 0 ; i < goukaku.length ; i++ ) {
```

```
  trace( goukaku[ i ].name + "¥t点数:" + goukaku[ i ].score );
}

//filter関数
function scoreFilter( student:Object, index:int, ary:Vector.<Object>
):Boolean {
  if ( student.score >= 60 ) return true;  //scoreが60以上の場合、結果に含める
  else return false;
}
```


Arrayクラスのfilter()メソッドのサンプルとほぼ同じ処理をVectorで行っています。異なる点は要素のデータ型をすべての個所で決めていることです。そのため、記述がやや複雑になる印象を受けます。

▶ 実行結果

```
山田一郎 点数:88
鈴木二郎 点数:78
三輪彰夫 点数:63
```

4-2-3 ベクターの要素をソートする

ベクター内の要素をソートするには、関数を作成してソートのロジックを作成する必要があります。このようなソート処理を行うために、Vectorクラスのsort()メソッドが用意されています。このメソッドに設定する関数は2つのベクターのベース型引数を受けて数値を返す必要があります。

Vectorクラスのソートメソッド詳細

メソッド	メソッドの詳細
`sort( compareFunction:Function, ):Vector.<T>`	**compareFunction**：ソートのロジックを決定する関数です。2つの引数を取り、数値を返す関数です。

Vectorクラスのsort()メソッドに指定する関数の書式

関数	関数の詳細
`function compare( item1:T, item2:T ):Number`	**item1**：ベクターのベース型オブジェクトです。 **item2**：ベクターのベース型オブジェクトです。 item1とitem2を使用してソート順を決定するロジックを書きます。 item1がitem2の前にソートされるケースのロジックでは、マイナスの数値を返し、その逆のロジックではプラスの数値を返すようにします。どちらでもない（等しい）ケースの場合、0を返します。 例えばベース型にscoreという数値フィールドがあり、その昇順でソートしたい場合、次のようなロジックを書きます。 if (item1.score < item2.score) return 負の数値; if (item1.score > item2.score) return 正の数値; return 0;

SAMPLE CODE

Vectorクラスのfilter()メソッドを使う

VectorSampleC.fla

```
//ベクターオブジェクトの作成
var students:Vector.<Object> = new Vector.<Object>();
students.push( { name:"山田一郎", score: 88 } );
students.push( { name:"鈴木二郎", score: 78 } );
students.push( { name:"須藤俊一", score: 25 } );
students.push( { name:"宮田三郎", score: 44 } );
students.push( { name:"三輪彰夫", score: 63 } );
//ソートと結果の出力
```

```
students.sort( compare );
for ( var i:int = 0 ; i < students.length ; i++ ) {
  trace( students[ i ].name + "¥t点数:" + students[ i ].score );
}
//ソートロジックを決定する関数
function compare( s1:Object, s2:Object ):Number {
  if ( Number( s1.score ) < Number( s2.score ) ) return -1; //s1の
  scoreが小さい場合、s1はs2の前にソートされます。
  if ( Number( s1.score ) > Number( s2.score ) ) return 1; //s1の
  scoreが大きい場合、s1はs2の後ろのソートされます。
  return 0;          //scoreが等しい場合ソートはしません。
}
```

①がソート関数です。scoreの昇順でソートしたい場合、このような記述で実現することができます。

▶ 実行結果

```
須藤俊一 点数:25
宮田三郎 点数:44
三輪彰夫 点数:63
鈴木二郎 点数:78
山田一郎 点数:88
```

PART 5

テキストと正規表現処理

5－1－1 ストリングオブジェクト（String）を使う

ストリングは一連の文字からなるデータで、「文字列」や「テキスト」とも呼ばれます。ActionScriptプログラム中に " "（ダブルクオーテーション）または ' '（シングルクオーテーション）で囲まれた場所はストリングデータとして認識されます。ストリングを変数に格納して処理するときはStringクラスを使用します。Stringクラスを使用することで、プロパティとメソッドを使用して下表のような処理を行うことができます。

■ Stringクラスの主なプロパティ

目的	プロパティ名	プロパティの概要
ストリングの文字数を数える	`length: int`	Stringオブジェクトに格納されているテキストの文字数

■ Stringクラスの主なメソッド

目的	メソッド名	メソッドの概要
ストリング内の1文字を取り出す	`charAt`	ストリング中の指定した位置の1文字を取り出します。
ストリング内の文字列を取り出す	`slice`	ストリング中の指定した開始位置から終了位置までの文字列を取り出します。
	`substr`	ストリング中の指定した開始位置から指定文字数の文字列を取り出します。
	`substring`	ストリング中の指定した開始位置から終了位置までの文字列を取り出します。
ストリング内の文字列を検索する	`indexOf`	ストリングの先頭から指定した文字列を検索します。
	`lastIndexOf`	ストリングの末尾から指定した文字列を検索します。
	`match`	ストリングにある、指定した文字列パターンと一致するストリングをすべて取り出します。
	`search`	ストリングの先頭から指定した文字列を検索します。
ストリング内の文字列を置き換える	`replace`	ストリングの先頭から指定した文字列を検索して置換します。
ストリング内の大文字を小文字に変換する	`toLowerCase`	ストリングに含まれる大文字を小文字にします。
ストリング内の小文字を大文字に変換する	`toUpperCase`	ストリングに含まれる大文字を小文字にします。
他のストリングと連結する	`concat`	2つのストリングを連結します。

SAMPLE CODE

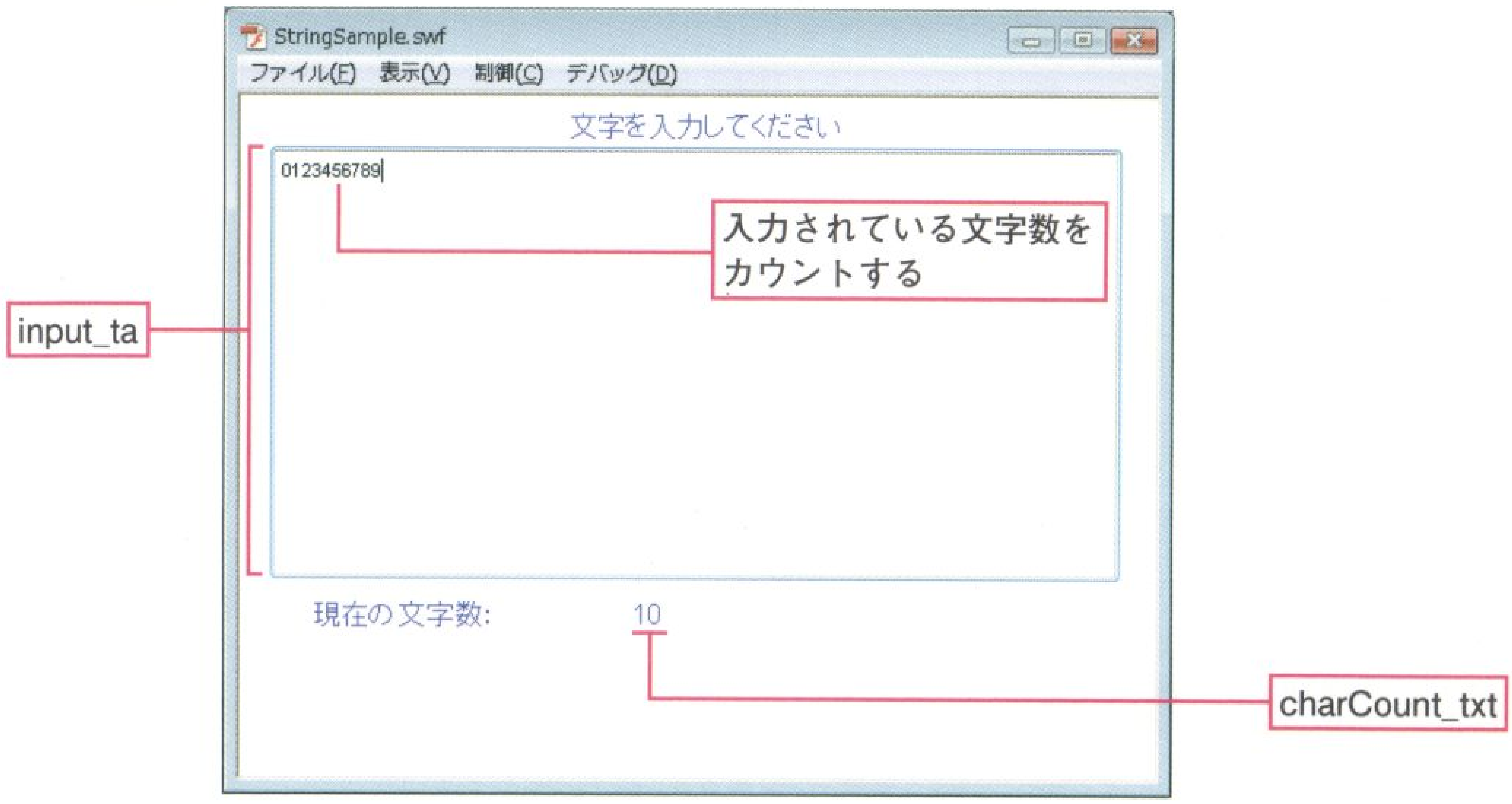
Stringオブジェクトを生成して文字数をカウントする

StringSample0.fla

```
var s:String = "ストリングの処理はStringクラスのオブジェクトで行います"; ――①
trace( "「" + s + "」の文字数は " + s.length + "あります。" ); ――②
var splus:String = s.concat( "。" ); ――③
trace( "「" + splus + "」の文字数は " + splus.length + "あります。" );
```

①でストリングオブジェクトを作成し、②でlengthプロパティにより、文字数を取得しています。③ではconcatメソッドにより文字列を追加しています。

SAMPLE



SAMPLE CODE

TextAreaコンポーネントに入力されている文字数をカウントする

StringSampleA.fla

```
function getCharCount( event:Event ):void {
  charCount_txt.text = input_ta.text.length.toString(); ――④
}
this.input_ta.addEventListener( Event.CHANGE,  getCharCount ); ――⑤
```

TextAreaに入力されているテキストはtextプロパティで取得できます。このプロパティはStringオブジェクトなので、④で行っているように、そのままlengthプロパティを参照できます。TextAreaコンポーネント(input_ta)に入力があると、Event.CHANGEイベントが発生するので、⑤でイベントリスナーの登録を行っています。

5-1-2 ストリングから一部の文字列を取り出す

ストリングオブジェクト内の一部の文字列を取り出す場合、取り出す文字列の開始位置と終了位置を指定します。各メソッドはStringオブジェクトから取り出した文字列を返しますが、オブジェクト内のテキストは変更されません。

■ Stringクラスの文字列を取り出すためのメソッド詳細

メソッド	メソッドの戻り値
charAt(index:Number = 0):String	indexで指定した位置の1文字を返します。先頭文字の位置は0です。
slice(startIndex:Number = 0, endIndex:Number = 0x7fffffff):String	startIndexの位置からendIndexの前までの文字列を返します。
substr(startIndex:Number = 0, len:Number = 0x7fffffff):String	startIndexの位置からlenで指定した文字数を含む文字列を返します。
substring(startIndex:Number = 0, endIndex:Number = 0x7fffffff):String	startIndexの位置からendIndexの前までの文字列を返します。

SAMPLE CODE

Stringオブジェクトから部分文字列を取り出すメソッドを実行する　StringSampleB.fla

```
var s:String = "文字列の検索処理はStringクラスに用意されたメソッドで行います。";
trace( s );
trace( "charAt( 9 ) = " + s.charAt( 9 ) );
trace( "slice( 4, 8 ) = " + s.slice( 4, 8 ) );
trace( "substr( 9, 5 ) = " + s.substr( 9, 6 ) );
trace( "substring( 4, 8 ) = " + s.substring( 4, 8 ) );
trace( "パラメータの省略例" );
trace( "slice( 4, ) = " + s.slice( 4 ) ); ―――――― ①
trace( "substr( 9 ) = " + s.substr( 9 ) );
trace( "substring( 4 ) = " + s.substring( 4 ) );
```

①以降は第2パラメータを省略しています。その場合、「ストリングの末尾まで」という指定と同様になります。

5-1-3 ストリング中の文字列を検索する

ストリングオブジェクト内の検索を行う場合、見つかった対象の位置を調べる方法（ indexOf, lastIndexOf, search）と、パターンに一致した文字列のリストを配列で取得する方法（match）があります。見つかった位置は、先頭文字の位置を0とするint値で返されます。match()メソッドは見つかった要素を配列で返します。

String クラスの文字列を検索するためのメソッド詳細

メソッド	メソッドの戻り値
indexOf(val:String, startIndex:Number = 0):int	startIndex以降でvalで指定した文字列を検索して見つかった位置。先頭文字の位置は0です。見つからない場合は-1。
lastIndexOf(val:String, startIndex:Number = 0x7fffffff):int	ストリングを末尾から検索します。startIndex以前でvalで指定した文字列を検索して最初に見つかった位置。先頭文字の位置は0です。見つからない場合は-1。
match(pattern:*):Array	patternで指定した文字列と一致するストリングをすべて格納した配列。patternは正規表現も使用できます。一致がなかった場合、null。
search(pattern:*):int	patternで指定した文字列と一致した部分の先頭位置。patternは正規表現も使用できます。一致がなかった場合、-1。

SAMPLE CODE

String オブジェクト内の文字列を検索する　StringSampleC.fla

```
var s:String = "各メソッドで共通することは、メソッドはStringオブジェクトから取り出した文字列を返すことです。";
trace( s );
trace( "indexOf( 'メソッド' ) = " + s.indexOf( 'メソッド' ) );
trace( "lastIndexOf( 'メソッド' ) = " + s.lastIndexOf( 'メソッド' ) );
trace( "match( 'メソッド' ) = " + s.match( 'メソッド' ) );
trace( "search( 'メソッド' ) = " + s.search( 'メソッド' ) );
```

それぞれのメソッド呼び出しで"メソッド"という文字列を検索して見つかった位置、またはその文字列自体を返します。

5-1-4 ストリング中の文字列を置換する

PART
1
2
3
4
5
6
7
8
9

ストリングオブジェクト内の置換はreplaceメソッドで行います。置換する文字列の検索方法は、完全一致による指定と正規表現によるパターンマッチングが行えます。見つかった文字を置換するときに利用できる"$"によるオプション書式があります。

String クラスの文字列を取り出すためのメソッド詳細

メソッド	メソッドの戻り値
replace(pattern:*, repl:Object):String	patternで指定した文字列と一致する最初の文字列をreplで指定される文字列に置き換えた後のストリング全体。replには関数も指定できるため、Object型になっています。

$コードによるオプション書式

$ コード	置換内容
$$	見つかった文字列を「$」に置き換えるときに使用します。
$&	見つかった文字列を表します。
$`	見つかった文字列より前の部分文字列を表します。
$'	見つかった文字列より後ろの部分文字列を表します。
$ n	パターンで( )によるグループ指定をしているとき、一致した文字列をグループのインデックスにより取得できます。先頭グループのインデックスは1です。9まで指定できます。
$ nn	パターンで( )によるグループ指定をしているとき、一致した文字列をグループのインデックスにより取得できます。先頭グループのインデックスは01です。99まで指定できます。

SAMPLE CODE

完全一致で最初に見つかった文字列を置換する StringSampleD1.fla

```
var s:String = "各MethodでMethodは文字列を返します。";
trace( s.replace( 'Method', 'メソッド' ) );// 各メソッドでMethodは文字列を返します。
```

ストリングにはMethodという文字列が2つありますが、完全一致では最初の対象のみが置換されます。

▶ 実行結果

```
各メソッドでMethodは文字列を返します。
```

SAMPLE CODE

完全一致で見つかったすべての文字列を一括置換する StringSampleD2.fla

```
var s:String = "各MethodでMethodは文字列を返します。";
trace( s.replace( /Method/g, 'メソッド' ) );   // 各メソッドでメソッドは文字列を返
します。 ①
```

完全一致で見つかったすべての対象を一括置換するには、①で指定しているように、/検索文字列/g という正規表現の指定を行います。gはglobalの意味で、すべてを検索します。

▶ 実行結果

```
各メソッドでメソッドは文字列を返します。
```


SAMPLE CODE

$コードの内容を確認する StringSampleD3.fla

```
var s:String = "012_Method_345";
trace( s );
//"Method"を"$"に置き換えます。
trace( s.replace( /Method/, "$$" ) ); ②
//"Method"を"Method should be Replaced"に置き換えます。
trace( s.replace( /Method/, "$& should be Replaced" ) ); ③
//"Method"を"Method"の前にある文字列に置き換えます。
trace( s.replace( /Method/, "$`" ) ); ④
//"Method"を"Method"の後ろにある文字列に置き換えます。
trace( s.replace( /Method/, "$'" ) ); ⑤
//"012_Method"を012と"M"にグループ化して検索し、それぞれ[]で囲います。
trace( s.replace( /([0-9]+)_([A-Z])/g, "[$1][$2]" ) ); ⑥
```

②の"$$"による指定では、マッチした"Method"が"$"に置換されています。
③の"$&"は、マッチした"Method"を表します。
④の"$`"は、マッチした"Method"の前にある文字列を表します。
⑤の"$'"は、マッチした"Method"の後ろにある文字列を表します。
⑥では、"$1"が1番目の([0-9]+)グループにマッチした文字列、"$2"が2番目の([A-Z])グループにマッチした文字列を表します。

5-1-5 大文字/小文字の変換を行う

toLowerCase、toUpperCaseメソッドにより、Stringオブジェクト内の文字列中にあるアルファベットの大文字/小文字の変換ができます。Stringオブジェクト内の文字変換を行い、結果を返しますが、元のオブジェクト内のテキストは変更されません。

Stringクラスの文字列を取り出すためのメソッド詳細

メソッド	メソッドの戻り値
toLowerCase():String	ストリングに含まれる変換可能な大文字を小文字にして返します。
toUpperCase():String	ストリングに含まれる変換可能な小文字を大文字にして返します。

SAMPLE CODE

Stringオブジェクト内の大文字<=>小文字変換を行う

StringSampleE.fla

```
var s:String = "toLowerCaseメソッドは大文字>小文字の変換をします。";
trace( s );
trace( "s.toLowerCase() = " + s.toLocaleLowerCase() );
trace( "s.toUpperCase() = " + s.toLocaleUpperCase() );
```

toLowerCaseメソッドとtoUpperCaseメソッドは主にアルファベットの大文字/小文字変換を行います。ストリング中の変換可能な文字のみが対象となります。

▶ 実行結果

```
toLowerCaseメソッドは大文字>小文字の変換をします。
s.toLowerCase() = tolowercaseメソッドは大文字>小文字の変換をします。
s.toUpperCase() = TOLOWERCASEメソッドは大文字>小文字の変換をします。
```

5-1-6 URLエンコード/デコードを行う

flash.utilsパッケージに含まれるescapeMultiByte関数でストリングをURLエンコード、同じくunescapeMultiByte関数でURLデコードを行えます。URLエンコード処理はエスケープ、デコード処理はアンエスケープとも呼ばれます。

■ flash.utilパッケージのURLエンコード/デコード関数

メソッド	メソッドの戻り値
escapeMultiByte(value:String):String	valueをURLエンコードした値を返します。
unescapeMultiByte(value:String):String	valueをURLデコードした値を返します。

SAMPLE CODE

URLエンコード/デコードを行う　StringSampleF.fla

```
import flash.utils.escapeMultiByte;
import flash.utils.unescapeMultiByte;
trace( escapeMultiByte( "flashでエスケープ" ) );
trace( unescapeMultiByte( "flash%E3%81%A7%E3%82%A8%E3%82%B9%E3%82%B1%
E3%83%BC%E3%83%97" ));
```

エンコードする文字列に半角英数字(ASCII文字)が含まれる場合、それらはエスケープされずそのまま残ります。

▶ 実行結果

```
flash%E3%81%A7%E3%82%A8%E3%82%B9%E3%82%B1%E3%83%BC%E3%83%97
flashでエスケープ
```

5 － 2 正規表現

5 - 2 - 1 正規表現で行えることを把握する

正規表現は文字列のパターンを表現する記法です。ActionScriptは正規表現をサポートするため、アプリケーションでユーザが入力した文字列が電子メールアドレス、郵便番号や電話番号などの書式と一致するかを調べたり、ストリング内で、ある文字列のパターンを検索、置換することができるようになります。ActionScript3.0では、ECMAScript Edition 3言語仕様(ECMA-262)の定義に基づく正規表現を実装しており、正規表現のリテラルもサポートされています。正規表現は下表のような利用方法があります。

■ 正規表現の使用例

目的	例
数字を検索する	10,123,2000などの任意の数字を検索する。
指定桁数の数字を検索する	11,22などの2桁の数値を検索する。
指定桁数の数字の組み合わせを検索する	2桁の数字または3桁の数字を検索する。
	2桁の数字+1文字(b)+3桁の数字の組み合わせを検索する。
日付が書式に合っているかチェックする 日付を検索する	nnnn年nn月nn日の形式に合致するか調べる。
正しい郵便番号かチェックする 郵便番号を検索する	nnn-nnnn形式の郵便番号形式に合致するか調べる。
正しい電話番号かチェックする 電話番号を検索する	2-4桁 + 3-4桁 + 4桁形式の電話番号に合致するか調べる。
正しいメールアドレスかチェックする メールアドレスを検索する	標準的なメールアドレスであるか調べる。
正しいURLかチェックする URLを検索する	正しいURLであるか調べる。
全角文字のみを検索する	Shift-JISテキスト内の全角文字を抜き出す。
指定したパターンの文字列を検索する 全文検索を行う	山田 + 1文字 からなる名前を検索する。

5-2-2 基本的な正規表現を行う

正規表現による処理を行うには、RegExpクラスのインスタンスを作成します。Stringクラスのmatch()、replace()、search()でも正規表現インスタンスを使用してパターンマッチング行うことができます。RegExpクラスのコンストラクタにより、指定の正規表現を持つオブジェクトを生成できます。また、ActionScriptは正規表現リテラルをサポートするので、コンストラクタを使用しなくてもプリミティブデータ型と同じようにインスタンスを生成できます。インスタンスを生成したら、RegExpクラスやStringクラスのメソッドで正規表現によるパターンマッチングを行うことができます。

RegExpクラスのコンストラクタ

メソッド	メソッドの働き
RegExp(re:String, flags:String) re　：正規表現のパターン flags：正規表現で使用されるフラグ	コンストラクタでは、パラメータで正規表現の本体であるパターンと、オプションのフラグを指定してインスタンスを生成できます。

SAMPLE CODE

RegExpクラスのインスタンスを生成する

```
//コンストラクタによるインスタンス生成
var myRegExp:RegExp = new RegExp( "山田", "g" );

//リテラルによるインスタンス生成
var myRegExp:RegExp = /山田/g;
```

RegExpクラスのメソッド

メソッド	メソッドの働きと戻り値
exec(str:String):Object str：正規表現を検索するストリング	正規表現による検索を行い、一致がない場合null、一致があった場合、一致した文字列とインデックスをプロパティとして持つオブジェクトを返します。 このメソッドは複数のマッチング結果を調べる場合などに使用できます。
test(str:String):Boolean str：正規表現を検索するストリング	正規表現による検索を行い、一致がない場合false、一致があった場合、trueを返します。 このメソッドはメールアドレスのフォーマットが正しいかをチェックする場合などに使用できます。

RegExpクラスの主なプロパティ

メソッド	メソッドの戻り値
lastIndex: Number	exec()メソッドとtest()メソッドによるチェックを開始する位置です。設定することにより、ストリング内のどの部分から検索を開始するかを決めることができます。

SAMPLE CODE

完全一致による基本的な正規表現検索　RegExpSampleA.fla

```
var str:String = "山田、鈴木、五島、島田、柳";
var regexp:RegExp = /島田/; ――①
var regexp2:RegExp = /吉田/; ――②
trace( regexp.test( str ) ); ――③
trace( regexp2.test( str ) ); ――④
```

①、②でRegExpオブジェクトを作成していますが、フラグを含まない基本的なものです。このように/ /で囲んだ内側に文字列を指定することで、その文字列との完全に一致する部分をストリングから検索できます。

①の正規表現では、「島田」という文字列にマッチする個所を検索します。③の結果はtrueになります。

②で作成したオブジェクトは「吉田」を検索しますが、strに含まれないため、④のtest()メソッドはfalseを返します。

▶ 実行結果

```
true
false
```

5-2-3 文字を検索する(メタ文字とフラグ)

正規表現では、例えば「"田"で終わる2文字の文字列」というようなパターン検索を行うことができます。このパターンには例えば「山**田**、島**田**、吉**田**」などがマッチングします。正規表現で任意の1文字を表すには、「. 」(ドット)を使用するため、上記の正規表現は「 /.田/ 」になります。「.」(ドット)のように文字単位の表現を行うものを「メタ文字」と呼びます。

ActionScript3.0正規表現のメタ文字

メタ文字	説明	正規表現例	パターン一致例
. (ドット)	任意の1文字に一致します。改行文字(¥n)にはsフラグを設定した場合のみ一致します。	/.AB./g	A1AB2 AAABBB ezAB@z
^ (キャレット)	ストリングの先頭に一致します。mフラグを設定した場合は、行の先頭にも一致します。	/^AZ/g	AZ1234 AZAZazAZ
$ (ドル記号)	ストリングの末尾に一致します。mフラグを設定した場合は、改行文字(¥n)の直前の位置にも一致します。	/AZ $/g	AZ1234AZ azAZ
* (アスタリスク)	直前のアイテムについて0回以上の繰り返しに一致します。	/1*a/g	1a12a12311a
+ (プラス)	直前のアイテムについて1回以上の繰り返しに一致します。	/1+a/g	1a12a12311a
? (疑問符)	直前のアイテムについて1回または1回だけ一致します。	/1?a/g	1a12a12311a
| (パイプ)	|の右側または左側のパターンに一致します。	/1.|12 /g	1a12a12311a
¥ (円記号)	特殊文字が持つ特別な意味を無効化します。	/a ¥//g	1a/1b/13a/
( )	正規表現の中でグループを定義します。グループは複数のパターンの組み合わせで繰り返しを指定するときに使用します。	/(1|2)a/g	1a12a12311a
[ ]	[]内に指定したいずれかの1文字に一致します。-(ハイフン)を使用して文字の範囲を次のように指定できます。 [12] 1か2のいずれかに一致 [1-9] 1～9のいずれかに一致 [1-9a-z] 1～9またはa～zのいずれかに一致	(¥d ¥D)*	1a_12b123c 4d
		/[12]a/g	1a1b2a2z3x3y4w4z
		/[1-4]z/g	1a1b2a2z3x3y4w4z
		/[1-2w-y]/g	1a1b2a2z3x3y4w4z

グループの使用

複数のパターン文字をグループとして繰り返しの指定などを行うことができます。例えば(¥d¥D)*は数字(¥d)と数字以外(¥D)の文字の組み合わせが0回以上繰り返すことを指示します。

ActionScript3.0正規表現のグループ

制御文字	説明	例	パターン一致例
( )	直前アイテムの0回以上繰り返しに一致します。	/1*a/g	11a 112a 122ca

フラグの使用

正規表現パターンのみの指定では、実際には最初に見つかった一致箇所で検索は終了します。上記のメタ文字表の一致例では、複数の一致個所を示していますが、これは正規表現に / gというフラグをつけているためです。gはglobalを表し、複数個所の一致を行う指示となります。フラグは正規表現パターンの末尾の/ の後ろに指定します。

```
/ 正規表現パターン / フラグ
```

ActionScript3.0正規表現のフラグ

フラグ	説明	例	パターン一致例
g(global)	複数の一致を行います。	/.2/g	A1B2C3a1b2c3
i(ignoreCase)	大文字と小文字を区別しないで一致を行います(デフォルトでは大文字と小文字は区別されます)。	/c3/ig	A1B2C3a1b2c3
m(multiline)	複数行のストリングで、^(キャレット)を各行頭、$(ドル記号)を各行末に一致させます。デフォルトでは、^はストリングの先頭、$はストリングの末尾にのみ一致します。	/^a./mig	A1B2C3 a1b2c3 Another Line
s(dotall)	.(ドット)を改行文字(¥n)にも一致させます。デフォルトでは一致しません。	/3./sg	A1B2C3 a1b2c3 Another Line
x(extended)	正規表現パターン内の空白文字を無視します。パターンを読みやすくするために使用します。	/ ^ a . /xmig ※/^a./migと同じです。	A1B2C3 a1b2c3 Another Line

SAMPLE CODE

メタ文字を使用する正規表現検索

RegExpSampleB.fla

```
var regexp:RegExp;
//メタ文字サンプル
```

```
var s01:String = "1a12a12311a";
// .
regexp = /1./g; trace( s01.match( regexp ) );
// ^
regexp = /^1a/g; trace( s01.match( regexp ) );
// $
regexp = /1a$/g; trace( s01.match( regexp ) );
// *
regexp = /1*a/g; trace( s01.match( regexp ) );
// +
regexp = /1+a/g; trace( s01.match( regexp ) );
// ?
regexp = /1?a/g; trace( s01.match( regexp ) );
// |
regexp = /1.|12/g; trace( s01.match( regexp ) );
// ( )
regexp = /(1|2)a/g; trace( s01.match( regexp ) );
// [ ]
regexp = /[12]a/g; trace( s01.match( regexp ) );
s01 = "1a/1b/13a/";
// ¥(円記号)
regexp = /a¥//g; trace( s01.match( regexp ) );
```

gフラグの結果を確認するために、Stringクラスのmatch()メソッドを使用しています。このメソッドは一致した結果を配列に格納して返すため、trace()により、一致した文字列が出力されます。

▶ 実行結果

```
1a,12,12,11
1a
1a
1a,a,11a
1a,11a
1a,a,1a
1a,12,12,11
1a,2a,1a
1a,2a,1a
a/,a/
```

SAMPLE CODE

フラグを使用する正規表現検索 RegExpSampleC.fla

```
//フラグ
var str:String = "A1B2C3a1b2c3";
// g      (global)
regexp =/.2/g; trace( str.match( regexp ) );
// i      (ignoreCase)
regexp =/c3/ig; trace( str.match( regexp ) );
str = "A1B2C3¥na1b2c3¥nAnother Line";
// m      (multiline)
regexp =/^a./mig; trace( str.match( regexp ) );
// s      (dotall)
regexp =/3./sg; trace( str.match( regexp ) );
// x      (extended)
regexp =/ ^ a . /xmig; trace( str.match( regexp ) );
```

各フラグの結果を確認するために、Stringクラスのmatch()メソッドを使用しています。このメソッドは一致した結果を配列に格納して返すため、trace()により、一致した文字列が出力されます。

▶ 実行結果

```
B2,b2
C3,c3
A1,a1,An
3,3
A1,a1,An
```

5-2-4 数字を検索する（メタシーケンス）

ストリング中の数字は文字の一部ですが、正規表現では数字と一致する数字パターンが使用でき、よく使用されます。 数字のパターンは電話番号や郵便番号、クレジットカードをはじめとして多くの検証や検索に必要になるためです。例えば ¥d という表現は0〜9のいずれかの数字に一致するパターンです。¥dのように¥で始まる表現はメタシーケンスと呼ばれます。

■ ActionScript3.0正規表現のメタシーケンス

メタシーケンス	説明	例	パターン一致例
¥d	0〜9の数字1文字に一致します。	/¥d+/g	1a_12b-123c 4d
¥D	数字以外の1文字に一致します。	/¥D+/g	1a_12b-123c 4d
¥w	a〜z、A〜Z、0〜9、_ に一致します。これらの文字は単語構成文字と呼ばれます。	/¥w+/g	1a_12b-123c 4d
¥W	単語構成文字以外に一致します。	/¥W+/g	1a_12b-123c 4d
¥b	単語構成文字と空白文字などの単語構成文字以外の文字との境界位置と一致します。ストリングの先端か末尾が単語構成文字の場合、それらとも一致します。	/¥b¥d/g	1a_12b-123c 4d
¥B	単語構成文字に挟まれた位置、または単語構成文字以外の文字に挟まれた位置と一致します。	/¥B¥D+/g	1a_12b-123c 4d
¥s	スペース、タブ、改行、復帰に一致します。これらの文字は空白文字と呼ばれます。	/¥s¥d/g	1a_12b-123c 4d
¥S	空白文字以外に一致します。	/¥S¥d/g	1a_12b-123c 4d
¥n	改行文字に一致します。	ストリング中の各文字位置で一致します。	
¥r	復帰文字に一致します。		
¥t	タブ文字に一致します。		
¥v	垂直フィード文字に一致します。		
¥xnn	16進数nnで示されるASCII文字1文字に一致します。	/¥x42/	ABCDEあいう
¥unnnn	16進数nnnnで示されるUnicode文字1文字に一致します。	/¥u3044/	ABCDEあいう

繰り返し表現の使用

上記メタシーケンス表の例で /¥d+/g のように+記号を使用していますが、これは「直前のアイテムを1回以上繰り返す」という繰り返し表現です。つまり、¥d+ は「1つ以上の数字」と一

致します。正規表現では、このような繰り返し制御文字がいくつか用意されています。

ActionScript3.0正規表現の繰り返し制御文字

制御文字	説明	例	パターン一致例
*	直前アイテムの0回以上繰り返しに一致します。	/1*a/g	11a 112a 122ca
+	直前アイテムの1回以上繰り返しに一致します。	/1+a/g	11a 112a 122ca
?	直前アイテムの0回または1回の繰り返しに一致します。	/1?a/g	11a 112a 122ca
{m} {m,} {m,n}	繰り返し回数または回数の範囲を指定します。 {m}の場合m回、 {m,}の場合m回以上、 {m,n}の場合m回以上n回以下 の繰り返しに一致します。	/ ¥d{2}/g	11a 112a 122ca
		/ ¥d{2,}/g	11a 112a 122ca
		/ ¥d{2,3}/g	11a 112a 122ca

SAMPLE CODE

メタシーケンスを使用する正規表現検索 RegExpSampleD.fla

```
//メタシーケンス
s01 = "1a_12b-123c 4d";
// ¥d   数字を検索
trace( s01.match( /¥d+/g ) );
trace( s01.match( /¥D+/g ) );
// ¥w   単語構成文字を検索
trace( s01.match( /¥w+/g ) );
trace( s01.match( /¥W+/g ) );
// ¥b 単語構成文字の境界
trace( s01.match( /¥b¥d/g ) );
trace( s01.match( /¥B¥D/g ) );
// ¥s 空白文字
trace( s01.match( /¥s¥d/g ) );
trace( s01.match( /¥S¥d/g ) );
```

▶ 実行結果

```
1,12,123,4
a_,b-,c ,d
1a_12b,123c,4d
-,
1,1,4
a,_,b,c,d
 4
_1,-1,23
```

SAMPLE CODE

繰り返し制御コードを使用する正規表現検索

RegExpSampleE.fla

```
//繰り返し制御
s01 = "11a 112a 122ca";
// * : 0回以上の繰り返し
trace( s01.match( /1*a/g ) );
// + : 1回以上の繰り返し
trace( s01.match( /1+a/g ) );
// ? : 0/1回の繰り返し
trace( s01.match( /1?a/g ) );
// { m }
trace( s01.match( /¥d{2}/g ) );
// { m,  }
trace( s01.match( /¥d{2,}/g ) );
// { m, n }
trace( s01.match( /¥d{2,3}/g ) );
```

▶ 実行結果

```
11a,a,a
11a
1a,a,a
11,11,12
11,112,122
11,112,122
```

SAMPLE CODE

数字+文字のパターン検索

RegExpSampleF.fla

```
var s01:String = "1a_12b123c 4d";
//2桁の数字を検索
trace( s01.match( /¥d{2}/ ) );          //12
//2桁の数字または3桁の数字を検索
trace( s01.match( /¥d{3}|¥d{2}/g ) ); //12,123
//2桁の数字+1文字(b)+3桁の数字の組み合わせを検索
trace( s01.match( /¥d{2}b¥d{3}/ ) );  //12b123
```

5-2-5 電話番号やメールアドレスのフォーマットチェックを行う

前節までに解説した正規表現を使用して電話番号やメールアドレスなどのフォーマットが正しいかチェックできます。ここでは、実用的なサンプルをいくつか紹介します。

■ ActionScript3.0正規表現によるフォーマットチェック

目的	日付
調べるフォーマットの内容	nnnn年nn月nn日の形式に合致するか調べる。
正規表現パターンの例	/ ¥d{4}年¥d{2}月¥d{2}日/
解説	¥d{4} は4桁の数字、¥d{2} は2桁の数字に一致します。

目的	郵便番号
調べるフォーマットの内容	nnn-nnnn形式の郵便番号形式に合致するか調べる。
正規表現パターンの例	/ ¥d{3}- ¥d{4}/
解説	¥d{3} は3桁の数字に一致します。

目的	電話番号
調べるフォーマットの内容	2-4桁 + 3-4桁 + 4桁形式の電話番号に合致するか調べる。
正規表現パターンの例	/ ¥d{2,4}- ¥d{3,4}- ¥d{4}/
解説	¥d{2,4} は2桁以上4桁以下の数字に一致します。

目的	メールアドレス
調べるフォーマットの内容	標準的なメールアドレスであるか調べる。
正規表現パターンの例	/([0-9a-zA-Z]+[-._+&])*[0-9a-zA-Z]+@([-0-9a-zA-Z]+[.])+[a-zA-Z]{2,6}/
解説	[0-9a-zA-Z]+ は英数字の1文字以上の繰り返しに一致します。 [-._+&] は[ ]内のいずれかの1文字に一致します。 ( )* は( )内をグループとしてその繰り返しが0以上ある場合に一致します。 @ はこの位置に"@"がある場合に一致します。 ( )+ は( )内をグループとしてその繰り返しが1以上ある場合に一致します。 [a-zA-Z]{2,6} はアルファベット文字が2以上6以下繰り返す場合に一致します。

目的	URL
調べるフォーマットの内容	正しいURLであるか調べる。
正規表現パターンの例	/http: ¥/ ¥/[¥d ¥w-_. ¥/~, $!*'();:@=&+]+/
解説	¥/ は「/」という文字がその位置にある場合に一致します。 ¥d は0- 9の数字に一致します。 ¥w はa ～ z、A ～ Z、0 ～ 9、_ に一致します。 [ ]+ は[ ]内の文字のいずれかが1つ以上繰り返す場合に一致します。

注意：上記の正規表現パターンは多くのケースで使用できる1つの例で、完全なフォーマットチェックを行うものではありません。

SAMPLE CODE

フォーマットチェックを行う正規表現サンプル RegExpSampleG.fla

```
//指定書式の検索
var s02:String = "後の2011年10月10日に〒170-0013に手紙を書き、090-1111-2222に電
話します。";
//日付(nnnn年nn月nn日の形式)
trace( s02.match( /¥d{4}年¥d{2}月¥d{2}日/ ) );
//郵便番号
trace( s02.match( /¥d{3}-¥d{4}/ ) );
//電話番号(2-4桁 + 3-4桁 + 4桁)
trace( s02.match( /¥d{2,4}-¥d{3,4}-¥d{4}/ ) );

//メールアドレスのチェック
var mail:String = "r_miya-ta@mail.websysd.com";
var badMail:String = "r_miya-ta.mail.websysd.com";
var email:RegExp = /([0-9a-zA-Z]+[-._+&])*[0-9a-zA-Z]+@([-0-9a-zA-
Z]+[.])+[a-zA-Z]{2,6}/; ──────①
trace( mail + " : " + email.test( mail ) );
trace( badMail + " : " + email.test( badMail ) );
//URLのチェック
var url:String = "http://www.websysd.com";
var badUrl:String = "ftp://www.it-school.org";
var Url:RegExp = /http:¥/¥/[¥d¥w-_.¥/~,$!*'();:@=&+]+/;
trace( url + " : " +  Url.test( url ) );
trace( badUrl + " : " + Url.test( badUrl ) );
```

①のemail:RegExpでは( )によるグループを作成しています。

▶ 実行結果

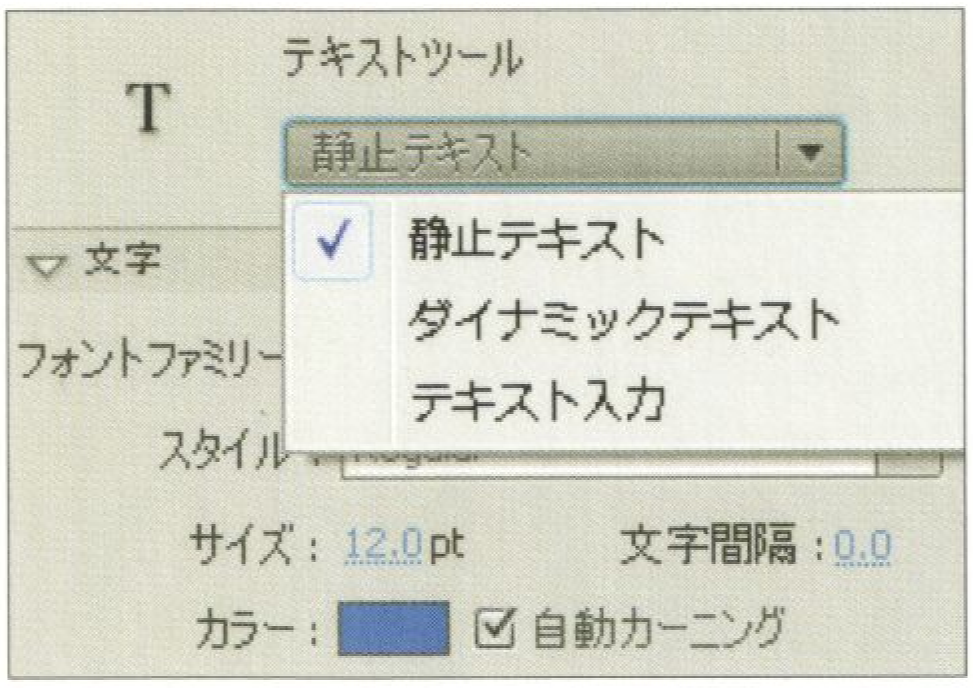
```
2011年10月10日
170-0013
090-1111-2222
r_miya-ta@mail.websysd.com : true
r_miya-ta.mail.websysd.com : false
http://www.websysd.com : true
ftp://www.it-school.org : false
```

2
5

5 - 3 - 1 テキストフィールドの種類とクラスを把握する

Flashのテキストツール(T)でステージ上に配置したテキストのうち、ダイナミックテキスト、またはテキスト入力として作成したオブジェクトはActionScriptでflash.text.TextFieldクラスのインスタンスとして操作できるようになります。これらのオブジェクトはActionScriptコードでも作成可能です。静止テキストとして作成したオブジェクトはActionScriptで操作することはできますが、このタイプのテキストインスタンスはActionScriptでは作成できません。

Flash CS5から、Flash Text Engine(FTE)をベースとするText Layout Framework(TLF)が導入されました。それに伴い、このPartで紹介する従来のテキストエンジンによる処理とコンポーネントは、「クラシックテキスト」と呼ばれるようになりました。TLFについては、Part6「高度なテキスト処理」をご覧ください。

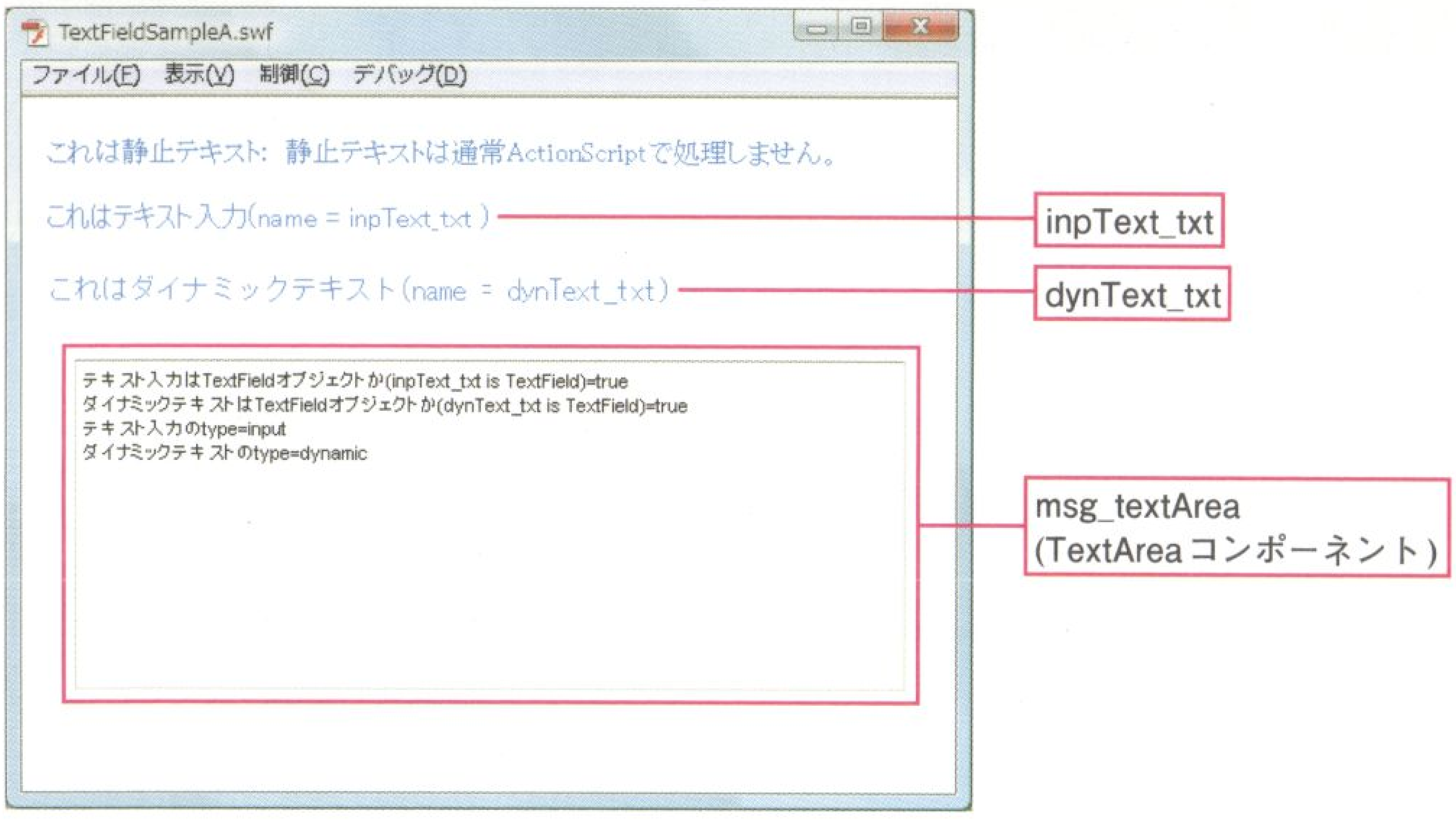


■ テキストツールで作成できるテキストの種類と関連するActionScript3.0クラス

種類	説明	クラス	typeプロパティ
静止テキスト	実行時に変更できない固定的なテキストになります。静止テキストはActionScriptでインスタンスを作成することができません。	StaticText	なし
ダイナミックテキスト	ユーザによるテキストの編集/入力はできませんが、ActionScriptで操作できるテキストになります。	TextField	dynamic
テキスト入力	ユーザが編集/入力でき、ActionScriptによる操作もできるテキストになります。		input

※typeプロパティはTextFieldクラスのもので、ダイナミックテキストかテキスト入力かを決定します。

SAMPLE



SAMPLE CODE

テキストフィールドの種類をActionScriptで調べる

TextFieldSampleA.fla

```
this.msg_textArea.text = "テキスト入力はTextFieldオブジェクトか
(inpText_txt is TextField)=" + (inpText_txt is TextField) ;
this.msg_textArea.text += "¥nダイナミックテキストはTextFieldオブジェクトか
(dynText_txt is TextField)=" + (dynText_txt is TextField) ;
this.msg_textArea.text +=
"¥nテキスト入力のtype=" +inpText_txt.type ;
this.msg_textArea.text +=
"¥nダイナミックテキストのtype=" + dynText_txt.type;
```

(①: 1～4行目、②: 5～8行目)

①では、画面上部の2つのテキスト(inpText_txtとdynText_txt)がTextFieldクラスのインスタンスかどうかをis演算子で判定し、②ではそれぞれのテキストのtypeプロパティの値を調べています。そして結果はmsg_textAreaというtextAreaコンポーネントに出力しています。

5-3-2 テキストフィールドオブジェクトを使う

PART
1
2
3
4
5
6
7
8
9

画面から入力されたテキスト内容を処理したり、テキストをActionScriptで作成して表示するには、TextFieldクラスを使用します。TextFieldクラスには次のようなプロパティが用意されており、ActionScriptで処理を行うことができます。

■ TextFieldクラスの主なプロパティ

目的	プロパティ名	プロパティの概要
自動的に拡大/縮小、整列する	autoSize: String	テキストフィールドの自動的なサイズ変更と整列を制御します。次の値をとることができます。 **TextFieldAutoSize.NONE**(デフォルト):サイズ変更を行いません。 **TextFieldAutoSize.LEFT**:左揃えテキストとして右マージンが伸縮します。 **TextFieldAutoSize.RIGHT**:右揃えテキストとして左マージンが伸縮します。 **TextFieldAutoSize.CENTER**:中央揃えテキストとして左右のマージンが伸縮します。
背景色を指定する	backgroundColor: uint	RGBカラー値によりテキストフィールドの背景色を設定します。デフォルトは白(0xFFFFFF)です。0xは16進数を表します。
境界線を表示/非表示にする	border:Boolean	**true** :境界線を表示します。 **false**:境界線を表示しません(デフォルト)。
境界線の色を指定する	borderColor:uint	RGBカラー値によりテキストフィールドの境界線色を設定します。デフォルトは黒(0x000000)です。0xは16進数を表します。
カーソルの位置を調べる	caretIndex:int	現在カーソルが置かれている位置を示します。
パスワード入力用にする	displayAs Password:Boolean	**true** :入力文字はアスタリスク表示されます。 **false**:入力文字はそのまま表示されます。
htmlを表示する	htmlText:String	基本的なタグによるHTMLを表示できます。
文字数を調べる	length:int	テキストフィールドの文字数を返します。
入力できる文字数を制限する	maxChars:int	テキストフィールドに入力できる最大文字数を設定できます。0の場合、文字数は無制限となります。
複数行を表示可能にする	multiline: Boolean	**true** :複数行のテキストを表示、編集可能です。 **false**:単一行の表示、編集を行います。

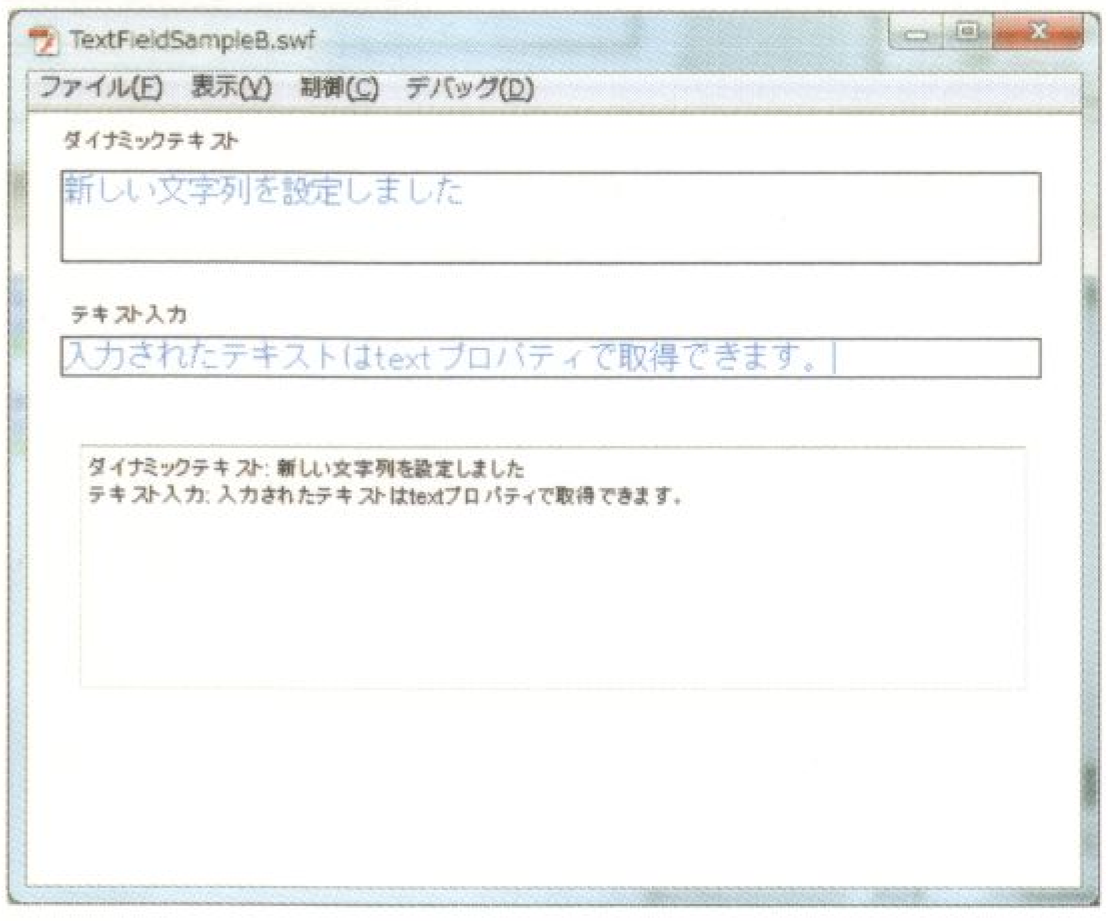
TextFieldクラスの主なプロパティ(続き)

目的	プロパティ名	プロパティの概要
文字を選択可能/不可能にする 文字のコピーを制限する	selectable:Boolean	**true** ：文字は選択可能になります。 **false**：文字を選択できなくなり、コピーできません。
入力できる文字を制限する	restrict:String	テキストフィールドに入力できる文字を設定されたもののみに限定します。
スタイルシートを関連づける	styleSheet:StyleSheet	テキストフィールドにスタイルが定義されたStyleSheetオブジェクトを関連づけてスタイルを利用可能にします。
文字列全体を取得/設定する	text:String	テキストフィールド内のプレーンテキストを取得できます。このプロパティで表示する文字を設定できます。
文字色設定/取得する	textColor:uint	RGBカラー値によりテキストの色を設定します。デフォルトは黒(0x000000)です。0xは16進数を表します。
テキストフィールドのタイプを取得/設定する	type:String	**TextFieldType.DYNAMIC**：ダイナミックテキストフィールドとして働きます。実際にセットされる値はdinamicです。 **TextFieldType:INPUT**：テキスト入力フィールドとして働きます。実際にセットされる値はinputです。
文字列の折り返しを行う	wordWrap:Boolean	**true**：文字列の幅がテキストフィールドより長い場合、折り返して表示します。 **false**：折り返ししません。

TextFieldクラスの主なメソッド

目的	メソッド名	メソッドの概要
文字列を追加する	appendText	テキストフィールドの末尾にテキストを追加します。
選択された文字列を置き換える	replaceSelectedText	現在選択されている範囲の文字列を置き換えます。
指定した範囲の文字列を置き換える	replaceText	開始位置と終了位置を指定して文字列を置き換えます。
範囲を指定して文字を選択する	setSelection	開始位置と終了位置を指定して文字列を選択します。

SAMPLE



SAMPLE CODE

ステージ上のダイナミックテキストに文字列を表示してテキスト入力に入力された文字列を取得する　TextFieldSampleB.fla

```
dynText_txt.addEventListener( MouseEvent.CLICK, setText ); ──①
function setText(event:Event):void { ──②
  dynText_txt.text = "新しい文字列を設定しました";
}
inpText_txt.addEventListener( Event.CHANGE, showText ); ──③
function showText(event:Event):void {
  output_txt.text = "ダイナミックテキスト: " + dynText_txt.text + "¥nテキスト入
  力: " + inpText_txt.text; ──④
}
```

①の設定でダイナミックテキスト(dynText_txt)がクリックされたときにsetText()関数を呼び出し、②の処理でそこで表示する文字列を更新します。

③の設定でinpText_txtに入力があるとshowText()関数が呼び出されます。④でtextプロパティにより、TextFieldオブジェクトからテキストを取得しています。

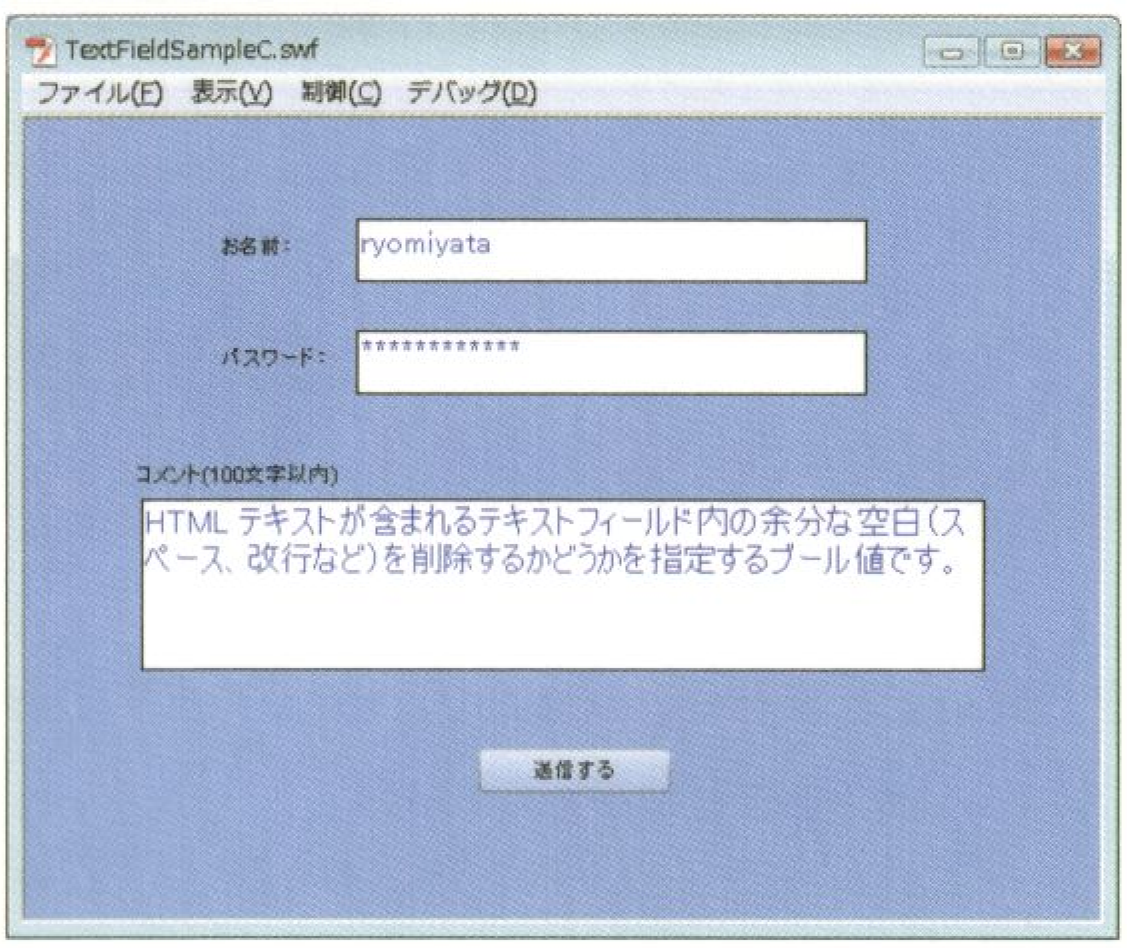
5-3-3 入力用のテキストフィールドを作る

TextFieldクラスのプロパティ設定により、入力用のテキストフィールドをアレンジできます。テキスト入力に関連するプロパティとその詳細は下表のようになります。

TextFieldクラスのテキスト入力に関するプロパティ詳細

プロパティ	詳細(用途)
border:Boolean	テキスト入力の境界線を表示します。入力ボックスの場合、境界線が必要となるため、通常はtrueにします。プロパティインスペクタでも設定できます。
displayAsPassword :Boolean	パスワード入力用のテキスト入力の場合、trueにして入力文字をアスタリスクで表示します。入力された文字列はtextプロパティで取得できます。
maxChars:int	テキスト入力に入力可能な最大文字数を指定できます。データベースに格納する場合などにフィールドのテキスト長を超えないようにできます。
multiline:Boolean	HTMLの<input type="text">のような単一行入力を行うときにはfalseに、<TextArea>のような複数行入力を行うときはtrueにします。
text:String	入力された文字列を取得できます。ActionScriptでこのプロパティにストリングをセットしてテキスト入力の文字列を設定できます。
type:String	ダイナミックテキストかテキスト入力かを決定します。テキスト入力の場合、"input"にします。この値はTextFieldTypeクラスにINPUTという定数でも定義されています。
wordWrap:Boolean	テキストエリアとして複数行の入力を行う場合、入力テキストがテキストフィールドの右端で折り返し改行させるには、trueに設定します。デフォルトはfalseなので、設定が必要です。プロパティインスペクタの「タイプ」で設定もできます。

SAMPLE



SAMPLE CODE

テキストフィールドのプロパティをActionScriptで設定する

TextFieldSampleC.fla

```
//お名前　テキスト入力の設定
name_txt.type = TextFieldType.INPUT;
name_txt.border = true;
name_txt.background = true;
name_txt.backgroundColor = 0xFFFFFF;

//パスワード　テキスト入力の設定
password_txt.type = TextFieldType.INPUT;
password_txt.border = true;
password_txt.displayAsPassword = true;
password_txt.background = true;
password_txt.backgroundColor = 0xFFFFFF;

//コメント　テキスト入力の設定
comment_txt.type = TextFieldType.INPUT;
comment_txt.border = true;
comment_txt.wordWrap = true;
comment_txt.maxChars = 100;
comment_txt.background = true;
comment_txt.backgroundColor = 0xFFFFFF;

//ボタンハンドラ関数
function sendMsg( event:MouseEvent ):void {
  trace( "お名前=" + name_txt.text + "¥nパスワード=" + password_txt.
  text +                                                              ①
    "¥nコメント= " + comment_txt.text + "(" + comment_txt.length +
    "文字)" );
}

//イベントリスナーの登録
send_btn.addEventListener( MouseEvent.CLICK, sendMsg );
```

　一部のプロパティはプロパティインスペクタで設定できますが、ActionScriptですべての設定を行っています。①で入力されたテキストとコメントの長さを取得しています。

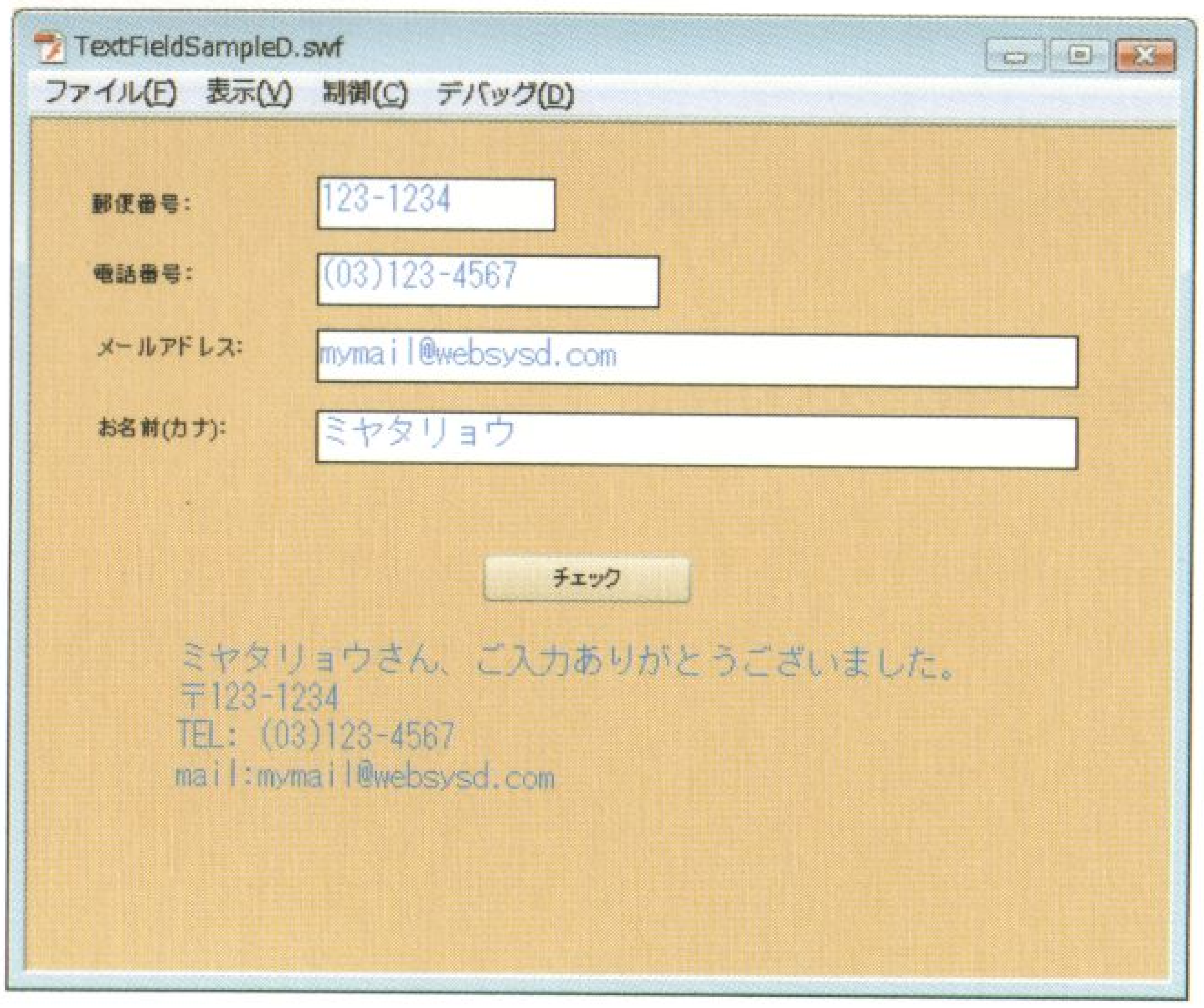
5-3-4 テキスト入力フィールドに入力できる文字を制限する

電話番号やメールアドレスなどのテキスト入力では、入力された文字列に不正な文字がないかチェックする必要があります。その場合、入力後にチェックを行うよりもTextFieldクラスのrestrictプロパティで入力フィールド自体に入力制限を設けることができます。

TextFieldクラスのテキスト入力文字を制限するプロパティ

プロパティ	詳細(用途)
restrict: String	「電話番号の場合は数字と括弧( )のみ」、「郵便番号の場合は数字とハイフン(-)のみ」というような入力文字の制限を行うことができます。文字の範囲をハイフン(-)で指定できます。カレット(^)で除外する文字を指定します。次のような設定ができます。 **"あいうえお"**：「あ」～「お」までの文字のみの入力を許可します。 **"あ-ん"**：ひらがなのみの入力を許可します。 **"あ-んア-ン"**：ひらがなとカタカナのみ入力できます。 **"0-9a-zA-Z()@"**：数字と英字、()@のみ入力できます。 **"あ-ん^た"**：「た」を除くひらがなを入力できます。 **"0-9¥¥-"**：0～9の数字とハイフン(-)を入力できます。ハイフン(-)は範囲を表す記号なので、¥¥をつけてエスケープする必要があります。その他、カレット(^)は¥¥^、円記号は¥¥¥¥という表記になります。

SAMPLE



SAMPLE CODE

TextField クラスのrestrictプロパティを使う

TextFieldSampleD.fla

```
//郵便番号の restrict設定
zip_txt.restrict = "0-9¥¥-"; ―①

//電話番号の restrict設定
phone_txt.restrict = "0-9()¥¥-"; ―①

//メールアドレスの restrict設定
mail_txt.restrict = "0-9a-zA-Z¥¥-._+&]@"; ―①

//カナの restrict設定
kana_txt.restrict = "ア-ン";

//テキストを取得するボタンハンドラ関数
function chk(event:MouseEvent):void {
  if ( kana_txt.length == 0 ) return;
  result_txt.text = kana_txt.text + "さん、ご入力ありがとうございました。¥n" +
  〒 + zip_txt.text + "¥n" +
  TEL:  + phone_txt.text + "¥n" +
  mail: + mail_txt.text;
}

//イベントリスナーの登録
chk_btn.addEventListener( MouseEvent.CLICK, chk );
```

①でハイフン(-)を入力するためのエスケープを行っています。

5-3-5 ActionScriptでテキストのフォーマットを設定する

TextFieldクラスではプロパティでテキストの色と背景色を設定できますが、フォントサイズやインデント、マージンのような詳細フォーマットはTextFormatクラスを使用するか、CSSを使用して設定できます。また、HTMLの基本タグを使用してフォーマットすることもできます。ここでは、TextFormatクラスを使用する方法を解説します。

ActionScriptでテキストのフォーマットを設定する方法

方法	概要
TextFormatクラス	ActionScriptでTextFormatオブジェクトを作成してTextFieldのdefaultTextFormatプロパティに設定するか、setTextFormat()メソッドに渡して設定します。プログラム実行中に動的にフォーマット設定を行うときに適しています。
CSS	HTMLで使用するCSSプロパティと同様のものをFlashで使用できます。StyleSheetオブジェクトを生成して動的にCSSスタイルを割り当てる方法と、外部CSSファイルを読み込む方法があります。
HTML	HTMLの基本タグによるテキストのフォーマットを行います。ActionScriptをあまり使用せずに手軽にテキストのフォーマット行うことができます。

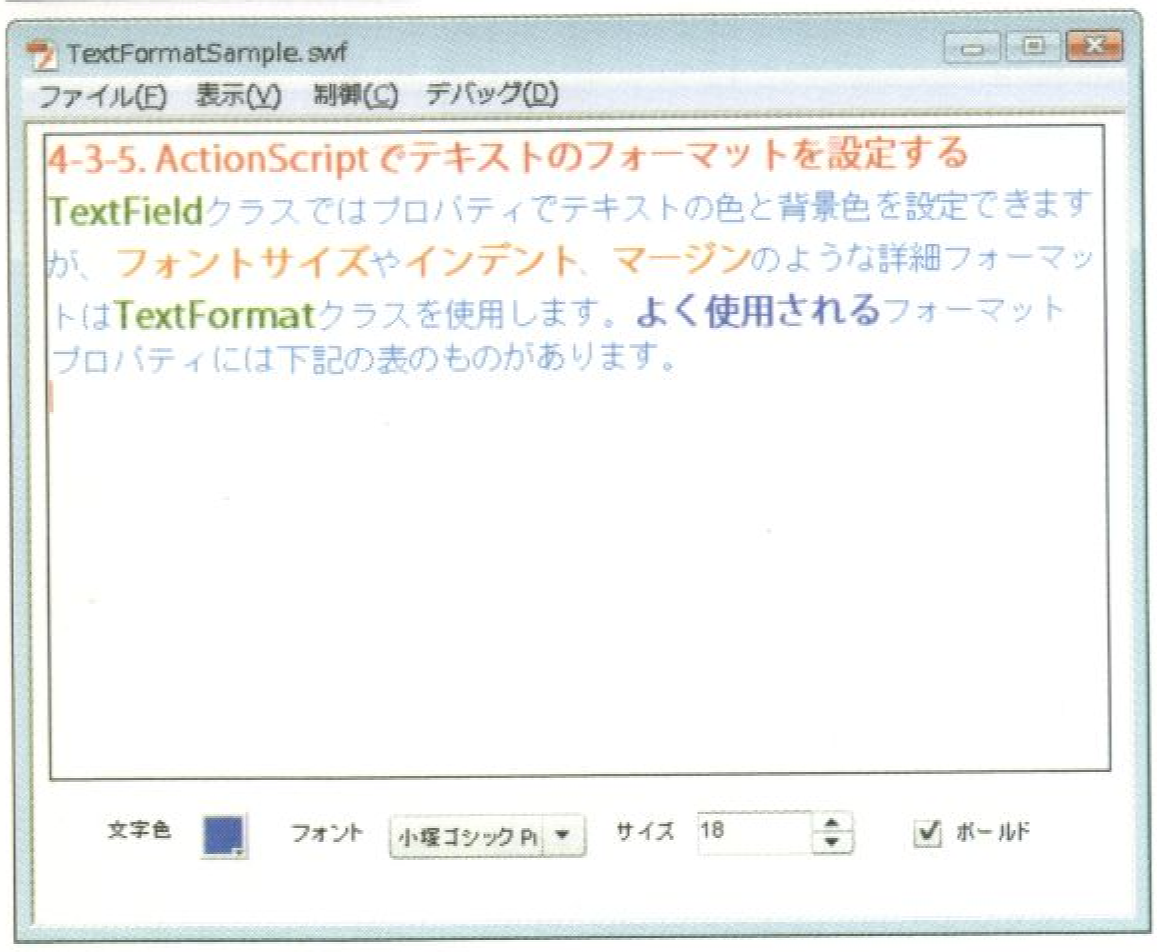
TextFormatクラスの主なプロパティ

目的	プロパティ名	プロパティの概要
段落のアラインメント設定	`align:String`	段落に左(デフォルト)、右、中央、均等割り付けを設定できます。段落は改行で区切られたブロックです。設定値はTextFormatAlignクラスに定数で定義されています。 **TextFormatAlign.LEFT**　：左寄せ **TextFormatAlign.RIGHT**　：右寄せ **TextFormatAlign.CENTER**：中央寄せ **TextFormatAlign.JUSTIFY**：均等割り付け
全体のインデント設定	`blockIndent:Object`	TextField全体のインデントをピクセル単位で設定できます。
テキストをボールドにする	`bold:Object`	trueの場合、テキストの書式をボールドにします。
テキストの色を変える	`color:Object`	RGBカラー値によりテキストの色を設定します。デフォルト値はnullですが、Flash Playerでは黒(0x000000)が使用されます。0xは16進数を表します。
フォントを指定する	`font:String`	フォント名を示す文字列を指定します。例えばデバイスフォントを使用するには、"_ゴシック"をこのプロパティに設定します。
段落先頭文字のインデント設定	`indent:Object`	段落の先頭文字の左マージンからのインデントを整数で設定します。

TextFormatクラスの主なプロパティ（続き）

目的	プロパティ名	プロパティの概要
テキストをイタリックにする	italic: Object	trueの場合、テキストの書式をイタリックにします。
カーニングを有効にする	kerning: Object	trueの場合、カーニングが有効になり、文字が読みやすくなります。ただし、埋め込みフォントにのみ適用されます。
段落の左マージン設定	leftMargin: Object	段落の左マージンをピクセル単位で設定します。
文字間隔の調整	letterSpacing: Object	すべての文字の間隔をピクセル単位で指定できます。小数点つきの数値を使用できます。
段落の右マージン設定	rightMargin: Object	段落の右マージンをピクセル単位で設定します。
テキストに下線をつける	underline: Object	trueの場合、テキストの書式を下線つきにします。

SAMPLE



SAMPLE CODE

TextFormatを使用するシンプルなテキストエディタ

TextFormatSample.fla

```
//スタイルを保存する変数
var myColor:uint = 0x000000;
var myFontName:String = "_ゴシック";
var myFontSize:Number = 18;
var myBold:Boolean = false;
//TextFormatオブジェクト
var myTextFormat:TextFormat = new TextFormat();
//TextFieldのプロパティ設定
```

```
editor_txt.multiline = true;
editor_txt.wordWrap = true;
//イベントリスナーの登録
colorPicker.addEventListener( Event.CHANGE, setColor );
fontCombo.addEventListener( Event.CHANGE, setFontName );
sizeNumericStepper.addEventListener( Event.CHANGE, setFontSize );
boldCheckBox.addEventListener( Event.CHANGE, setBold );
//イベントリスナー
function setColor(event:Event):void {
  myColor = colorPicker.selectedColor;applyFormat();
}
function setFontName(event:Event):void {
  myFontName = fontCombo.selectedItem.data;     applyFormat();
}
function setFontSize(event:Event):void {
  myFontSize = sizeNumericStepper.value;applyFormat();
}
function setBold(event:Event):void {
  myBold = boldCheckBox.selected;applyFormat();
}
//スタイルを設定する関数
function applyFormat():void {
  if ( editor_txt.selectionBeginIndex == editor_txt.
    selectionEndIndex ||
    editor_txt.selectionBeginIndex >  editor_txt.length ||        ─①
    editor_txt.selectionEndIndex == 0 ||
    editor_txt.selectionEndIndex > editor_txt.length ) return;
  myTextFormat.color = myColor;
  myTextFormat.font = myFontName;
  myTextFormat.size = myFontSize;
  myTextFormat.bold = myBold;                                     ─②
  editor_txt.setTextFormat( myTextFormat, editor_txt.
  selectionBeginIndex, editor_txt.selectionEndIndex );
}
```

①では、選択範囲がなかったときなどに処理を中断しています。②でTextFormatオブジェクトのプロパティを設定してTextFieldの選択範囲に割り当てを行っています。

5-3-6 HTMLテキストを表示する

TextFieldオブジェクトは基本的なHTMLタグをサポートしており、HTMLによる書式の設定を行うこともできます。サポートされるタグと属性は下表のとおりです。

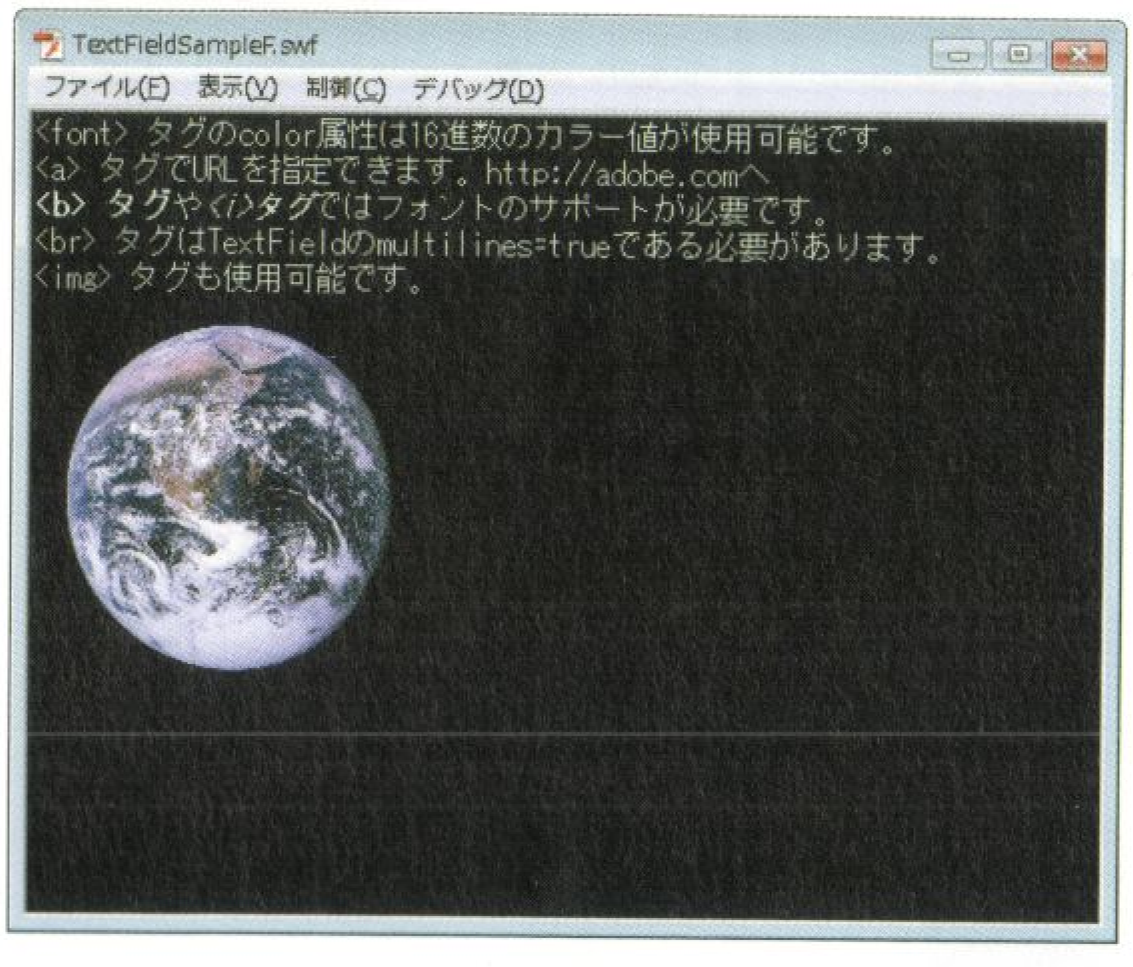
ActionScript3.0のTextFieldオブジェクトでサポートされるHTMLタグ

タグ	サポートされる属性と必要事項
<a>	**target** ：_self、_blank、_parent、_topを指定できます。 **href** ：URLとActionScript linkイベントを指定できます。
<b>	フォントがボールド書体をサポートしている必要があります。
 	TextFieldのmultilinesプロパティがtrueである必要があります。
<font>	**color** ：16進数のカラー(#FFFFFF形式)値のみ使用可能です。 **face** ：フォント名を指定します。カンマで区切られたフォント名の一覧を指定した場合、Flash Playerは最初に利用可能なフォントを選択します。指定したフォントがローカルにインストールされていないか、組み込まれていない場合、代替フォントが選択されます。 **size** ：フォントのサイズを整数で指定します。絶対ピクセルサイズ(12など)、相対ポイントサイズ(+1など)を使用できます。
<img>	テキストは、埋め込んだイメージの周囲に自動配置されるため、TextFieldのmultilinesプロパティがtrueである必要があります。 **src** ：イメージファイル、SWFファイルのURL、ライブラリ内のムービークリップシンボルへのリンケージ識別子を指定します。 **width** ：ピクセル単位の幅を指定します。 **height** ：ピクセル単位の高さを指定します。 **align** ：イメージの水平整列をleft(デフォルト)、rightで指定します。 **hspace** ：イメージ左右のテキストが表示されない領域を指定します。デフォルト値は8です。 **vspace** ：イメージ上下のテキストが表示されない領域を指定します。デフォルト値は8です。 **id** ：ActionScriptを使用してイメージオブジェクトを処理するための名前です。 **checkPolicyFile** ：trueの場合、クロスドメインポリシーファイル(crossdomain.xml)をFlash Playerが確認します。
<i>	フォントがイタリック体をサポートしている必要があります。
	、 は使用できません。
<p>	TextFieldのmultilinesプロパティがtrueである必要があります。 **align** ：left、right、justify、centerが使用可能です。 **class** ：flash.text.StyleSheetオブジェクトで定義されるCSSスタイルクラスを指定できます。
<span>	CSSテキストスタイルでのみ使用できます。 **class** ：flash.text.StyleSheetオブジェクトで定義されるCSSスタイルクラスを指定します。

■ ActionScript3.0のTextFieldオブジェクトでサポートされるHTMLタグ（続き）

タグ	サポートされる属性と必要事項
<textformat>	TextFormatクラスの段落フォーマットプロパティをhtmlで指定できます。 **blockindent** ：ブロックのインデントをポイント単位で指定します。 **indent** ：左マージンから段落の先頭文字までのインデントを指定します。 **leading** ：行間の行送り（垂直の行間隔）を指定します。 **leftmargin** ：段落の左マージンをポイント単位で指定します。 **rightmargin** ：段落の右マージンをポイント単位で指定します。 **tabstops** ：カスタムタブストップを負以外の整数の配列として指定します。
<u>	フォントがアンダーラインをサポートしている必要があります。

SAMPLE



SAMPLE CODE

基本的なHTMLテキストを表示する

```
html_txt.multiline = true;
html_txt.background = true;
html_txt.backgroundColor = 0x000000;
html_txt.htmlText =
<font color='#CCFFFF'>&lt;font&gt;タグのcolor属性は16進数のカラー値を使用可能です。
<br> +
<a href='http://adobe.com/'>&lt;a&gt; タグでURLを指定できます。http://adobe.
comへ</a><br> +
<b>&lt;b&gt; タグ</b>や<i>&lt;i&gt;タグ</i>ではフォントのサポートが必要です。<br> +
&lt;br&gt; タグはTextFieldのmultilines=trueである必要があります。<br> +
&lt;img&gt; タグも使用可能です。<br><img src='http://www.websysd.com/img/
itj/icons/earth.jpg' height='200' width='200'><br> +
</font>;
```

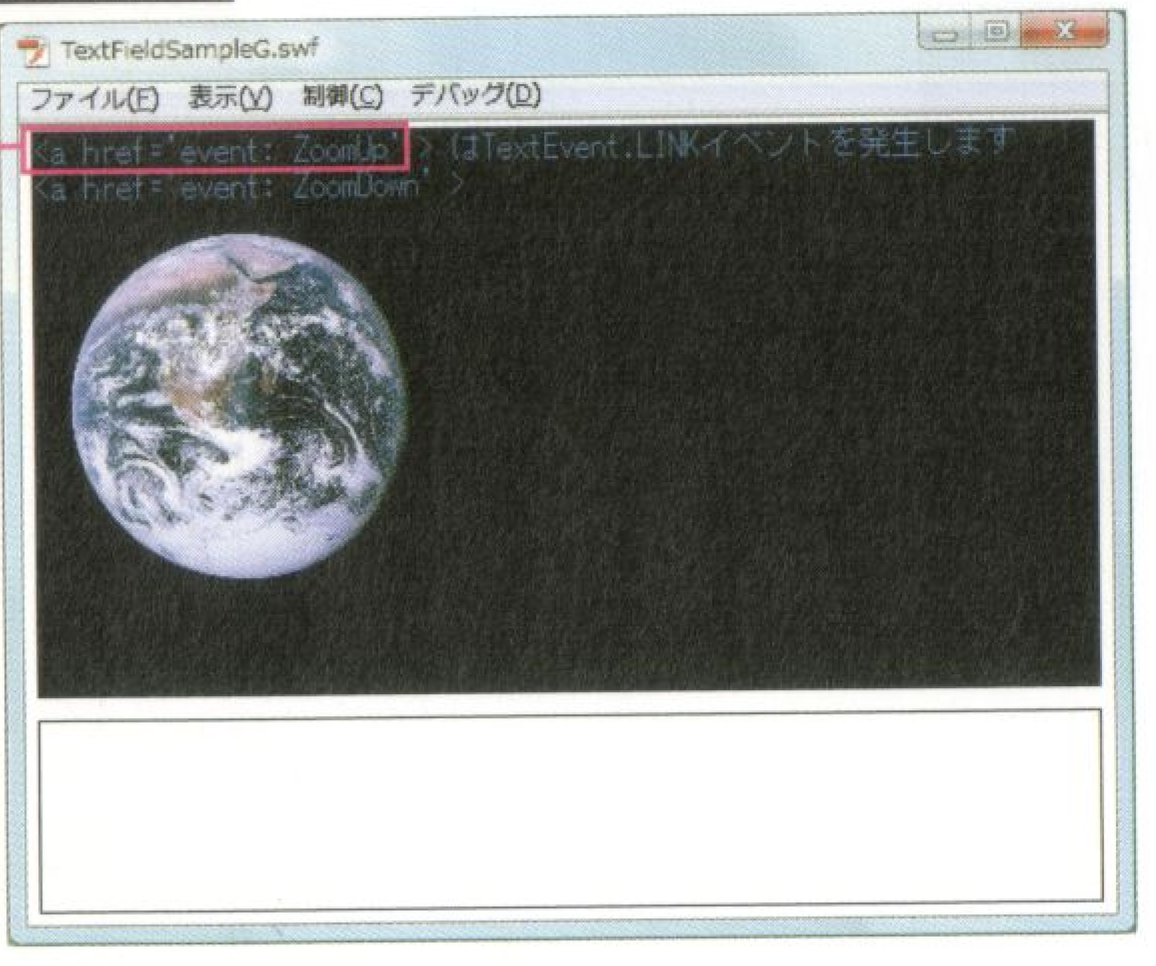
5-3-7 HTMLテキストのハイパーリンクでイベントを発生させる

TextFieldのhtmlTextプロパティに設定したhtmlテキストの<a>タグでActionScriptイベントを発生させることができます。イベントが発生した場合、イベントハンドラで処理できます。

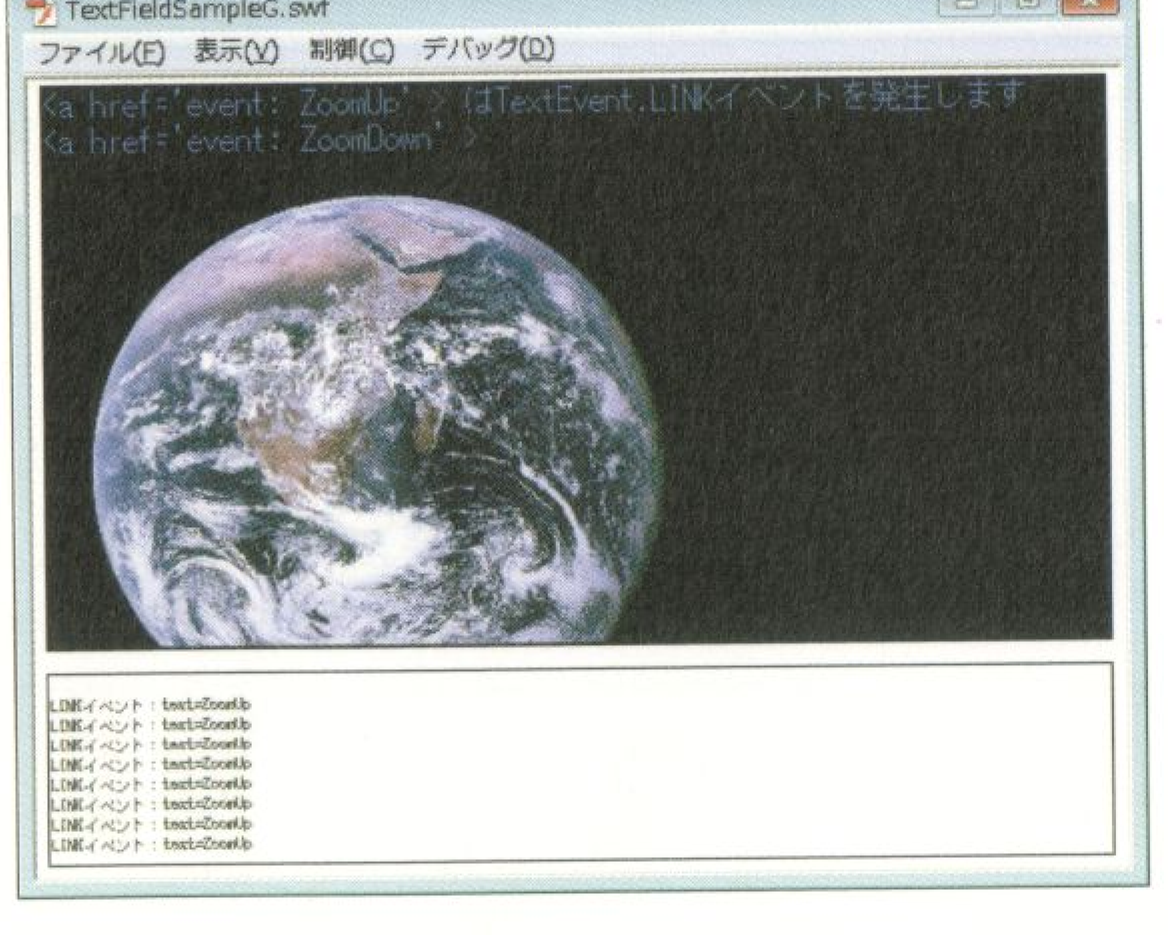
ActionScript3.0のTextFieldオブジェクトでイベントを発生させる<a>タグ

タグと属性	説明
<a href>	href="event:myText" の記述により、ハイパーリンクをクリックしたときにTextEvent.LINKイベントを発生できます。 myTextに指定したストリングは、ハンドラに渡されるTextEventイベントオブジェクトのtextプロパティで参照可能になります。 <a href = "event: AS3.html"> のリンクをクリックしたとき、TextEventオブジェクトのtextプロパティは"AS3.html"になります。

SAMPLE



■ <a href='event:ZoomUp'>部分を数回クリックした後の状態



SAMPLE CODE

HTMLテキストのハイパーリンクでイベントを発生させて処理する TextFieldSampleG.fla

```
var imgSize:uint = 200;  //imgタグの height、width属性に使用する値
html_txt.multiline = true;
html_txt.background = true;
html_txt.backgroundColor = 0x000000;
html_txt.selectable = false;
function setHtmlText():void {
  html_txt.htmlText =
  <a href='event:ZoomUp'>&lt;a href='event: ZoomUp' &gt; はTextEvent.
  LINKイベントを発生します</a><br> +
  <a href='event:ZoomDown'>&lt;a href='event: ZoomDown' &gt; </a><br>
  +
  <img src='http://www.websysd.com/img/itj/icons/earth.jpg' height='
  + imgSize + "' width='" + imgSize + "'>" +
  </font>;
}
function textLinkHandler( event:TextEvent ) { ──────────①
  msg_txt.appendText( "LINKイベント:text=" + event.text + "¥n" );
  switch ( event.text ) {
    case "ZoomUp":
      imgSize += 10;
      break;
    case "ZoomDown":
      if ( imgSize >= 10 ) imgSize -= 10;
      break;
  }
  setHtmlText();
}
html_txt.addEventListener( TextEvent.LINK, textLinkHandler );
setHtmlText();
```

<a href='event:ZoomUp'>をクリックすると"ZoomUp"が、<a href='event:ZoomDown'>をクリックすると"ZoomDown"がTextEvent.textプロパティに設定されて、TextEvent.LINKイベントが発生します。①から始まるイベントハンドラでこのtextプロパティを調べて変数imgSizeの値を更新します。この値を使用してimgタグのheight、width属性を設定しなおして再びhtmlTextにセットします。この操作により、TextFieldに表示されるイメージのサイズをズームアップ/ダウンさせます。

5-3-8 CSSを使用してテキストをフォーマットする

FlashではTextField内の文字列に対してCSSによるテキストフォーマットを適用できます。flash.text.StyleSheetクラスのインスタンスを作成して割り当てる方法と、外部のCSSファイルを読み込む方法があります。Flash PlayerでサポートされるCSSスタイルは下表のとおりです。スタイル名の規則として、CSSに対応するActionScriptプロパティ名は、ハイフンを含まないものになります。

StyleSheetクラス(Flash Player)でサポートされるCSSスタイル

目的	プロパティ名(CSS)	プロパティ名(ActionScript)	使用可能な値
フォント色	color	color	#FF00FFのような16進数のカラー値のみ "blue"などの名前付きカラーはサポートされません。
ボックスタイプ	display	display	inline、block、none
フォント名	font-family	fontFamily	任意のフォントファミリ名を優先度の高い順にカンマ区切りリストで指定できます。
フォントサイズ	font-size	fontSize	単位(px、pt)を除く値の数字の部分だけをピクセル値として指定します。
フォントスタイル	font-style	fontStyle	normal、italic
フォントウェイト	font-weight	fontWeight	normal、bold
カーニング	kerning	kerning	true、false 埋め込みフォントに対してのみサポートされています。カーニングはWindowsでのみ設定可能です。設定されたカーニングはWindows以外のバージョンのFlash Playerでも表示されます。
行間隔	Leading	leading	単位(px、pt)を除く値の数字の部分だけをピクセル値として指定します。
文字間隔	letter-spacing	letterSpacing	単位(px、pt)を除く値の数字の部分だけをピクセル値として指定します。
左マージン	margin-left	marginLeft	単位(px、pt)を除く値の数字の部分だけをピクセル値として指定します。
右マージン	margin-right	margin-right	単位(px、pt)を除く値の数字の部分だけをピクセル値として指定します。
テキストアライン	text-align	textAlign	left、center、right、justify
テキスト装飾	text-decoration	textDecoration	none、underline
インデント	text-indent	textIndent	単位(px、pt)を除く値の数字の部分だけをピクセル値として指定します。

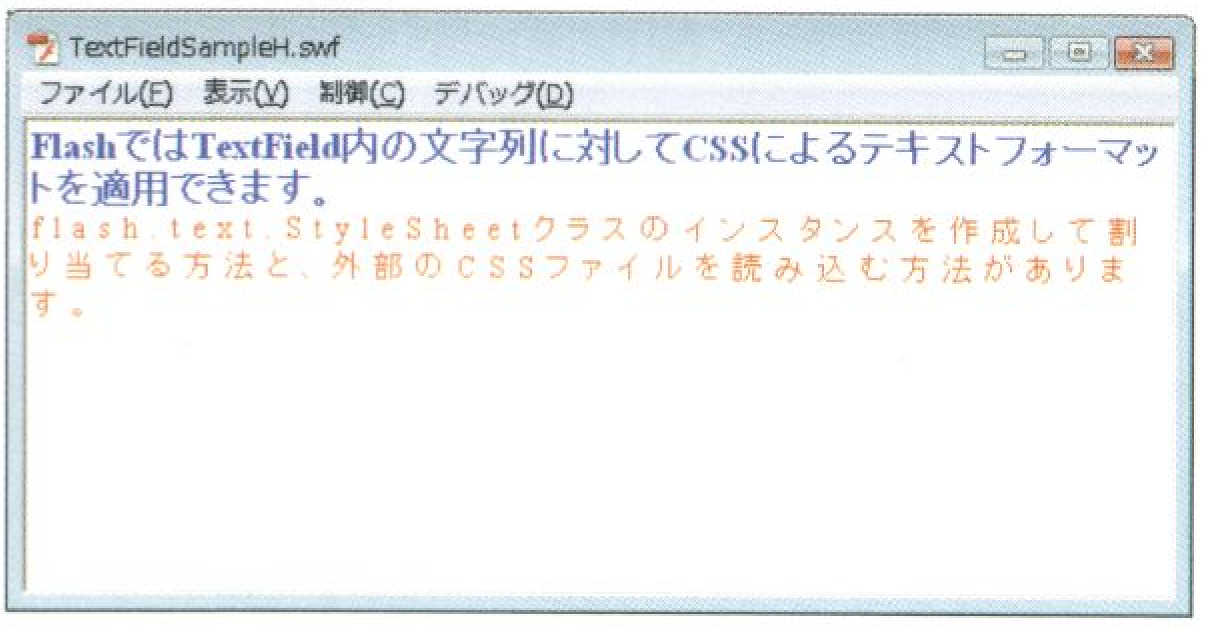
TextFieldクラスのスタイルシート設定のためのプロパティ詳細

プロパティ	詳細（用途）
styleSheet: StyleSheet	スタイルが定義されたStyleSheetオブジェクトをTextFieldオブジェクトに関連づけをします。このプロパティを変更して別のStyleSheetオブジェクトを割り当てるとTextFieldは再描画されて新しいスタイルが適用されます。
type: String	スタイルが設定されたTextField内のテキストは編集できません。そのため、typeプロパティはTextFieldType INPUTではなく、TextFieldType.DYNAMICにします。

TextFormatクラスのシート設定のためのメソッド詳細

メソッド	メソッドの詳細
setStyle (styleName:String, styleObject:Object) :void	動的にスタイルプロパティを設定したObjectオブジェクトをstyleObjectパラメータに指定し、styleNameで指定した名前のスタイルとして追加します。作成済みのスタイル名でメソッドを呼び出すと新しい値で上書きできます。 作成したスタイルを削除するには、styleObjectパラメータをnullで呼び出します。

SAMPLE



SAMPLE CODE

StyleSheetクラスのインスタンスを作成してスタイルを割り当てる

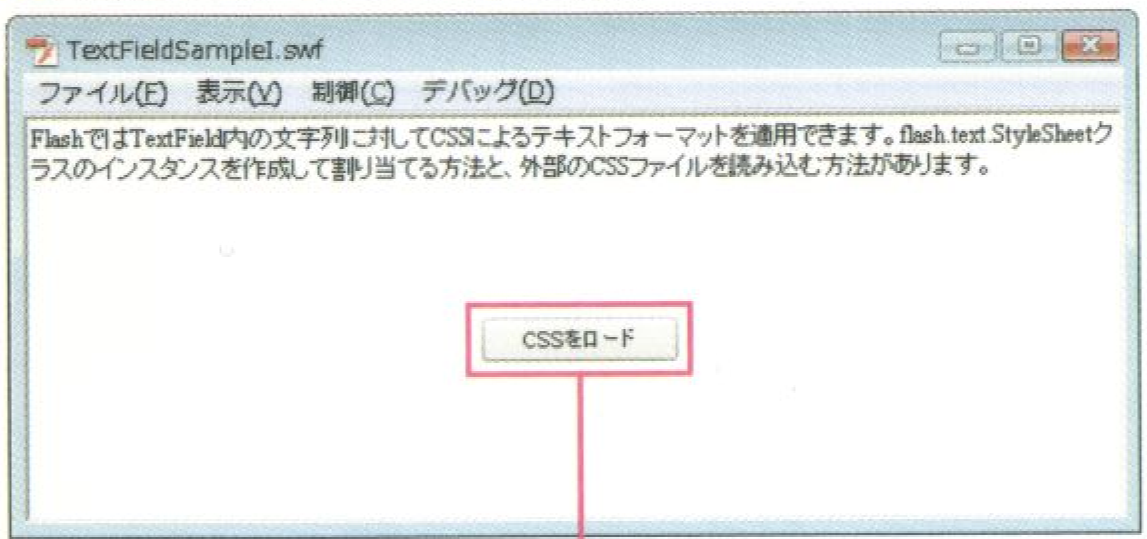
```
TextFieldSampleH.fla
//StyleSheetオブジェクトの生成
var style:StyleSheet = new StyleSheet();
//スタイル設定のためのオブジェクト生成(1)
var style1:Object = new Object();
style1.fontSize = 18;
style1.fontWeight = "bold";                          ①
style1.color = "#0000FF";
style.setStyle( ".bold_Blue_size18", style1 );
//スタイル設定のためのオブジェクト生成(2)
var style2:Object = new Object();                    ②
```

```
style2.fontSize = 14;
style2.letterSpacing = 6;
style2.leading = 2;                                          ②
style2.color = "#FF0000";
style.setStyle( ".Red_spc6_ld2_size14", style2 );
var tf:TextField = new TextField();
tf.width = 550;
tf.height = 220;
tf.multiline = true;
tf.wordWrap = true;
tf.styleSheet = style;
tf.htmlText = "<p class='bold_Blue_size18'>FlashではTextField内の文字列に対
してCSSによるテキストフォーマットを適用できます。</p>" +
"<p class='Red_spc6_ld2_size14'>flash.text.StyleSheetクラスのインスタンスを作
成して割り当てる方法と、外部のCSSファイルを読み込む方法があります。</p>";   ③
this.addChild(tf);
tf.x = 0;
tf.y = 0;
```

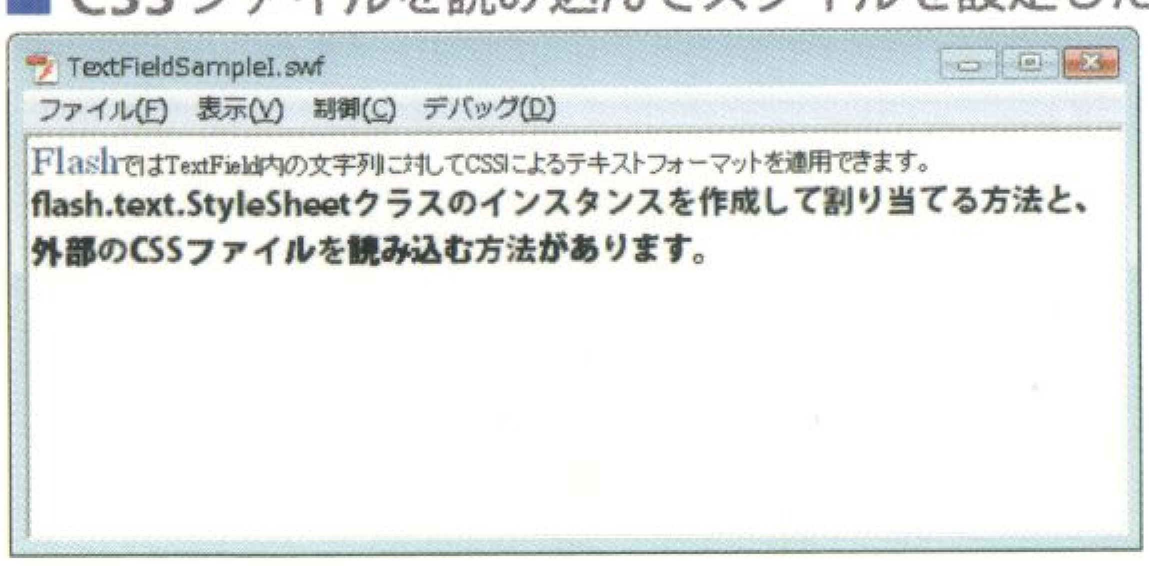
①で .bold_Blue_size18 という名前のクラスセレクタを定義しています。②で、.Red_spc6_ld2_size14 という名前のクラスセレクタを定義しています。③でhtmlTextを作成していますが、そこでクラスセレクタを使用しています。

SAMPLE

■ CSSファイル読み込み前



■ CSSファイルを読み込んでスタイルを設定した後



SAMPLE CODE

外部CSSファイルをロードしてスタイルを割り当てる

TextFieldSample1.css

```
p { ―――――――――――――――――――――――――――――――――――― ①
 font-family: 小塚ゴシック Std H, メイリオ, _ゴシック;
 font-size: 16;
}
.headLine { ――――――――――――――――――――――――――――――― ②
 font-size: 20;
 font-weighe: bold;
 color: #003366;
}
```


①ではすべての<p>タグに適用するタイプセレクタを定義しています。②から定義している.headLineスタイルはクラスセレクタです。

TextFieldSample1.fla

```
import fl.controls.Button;
var loader:URLLoader;
var tf:TextField;
var htmlText:String = "<span class='headLine'>Flash</span>では
TextField内の文字列に対してCSSによるテキストフォーマットを適用できます。" +
<p>flash.text.StyleSheetクラスのインスタンスを作成して割り当てる方法と、外部のCSSファイル
を読み込む方法があります。</p>;
//TextFieldオブジェクトを生成
tf = new TextField();
tf.width = 550;
tf.height = 200;
tf.multiline = true;
tf.wordWrap = true;
tf.htmlText = htmlText;
this.addChild( tf ); tf.x = 0; tf.y = 0;
//外部CSSのロードを開始するためのボタンを作成して配置
var btn:Button = new Button();                          ┐
btn.addEventListener(MouseEvent.CLICK, loadCss);         │
btn.label = "CSSをロード";                               │
this.addChild( btn );                                    ├― ③
btn.x = ( stage.width - btn.width ) / 2;                 │
trace( btn.x + "," + stage.width + "," + btn.width );    │
btn.y = 100;                                             ┘
```

```
//外部CSSファイルをロード
function loadCss(event:MouseEvent):void {                              ④
  var urlReq:URLRequest = new URLRequest( "TextFieldSampleI.css" );
  loader = new URLLoader();
  loader.addEventListener(Event.COMPLETE, cssLoadFinished );
  loader.load( urlReq );
}
//CSSファイルのロードが終了したときに呼び出されるハンドラ
function cssLoadFinished(event:Event):void { ──────────────────── ⑤
  //StyleSheetオブジェクトを生成してロードしたCSSを読み込む
  var style:StyleSheet = new StyleSheet();
  style.parseCSS( loader.data );
  tf.styleSheet = style;
  tf.htmlText = htmlText;
  removeChild( btn );
}
```

このサンプルでは、外部CSSの読み込みによりスタイルが動的に適用されることを確認するために、ボタンによる読み込みを行っています。③でボタンを作成して配置しています。④で外部CSSファイルをロードしていますが、ロードが完了した時点で⑤のcssLoadFinished関数が呼び出され、StyleSheetオブジェクトにロードCSSを読み込んでTextFieldに適用します。

5-3-9 ユーザ環境のフォント情報を調べる

Flashでユーザが利用できるフォントの情報を取得するには、flash.text.Fontクラスのメソッドとプロパティを使用します。FontクラスのenumerateFonts()メソッドにより、埋め込みのみ、または埋め込みとデバイスフォントのリストを作成できます。作成されるリストはFontオブジェクトの配列なので、Fontクラスのプロパティでフォント名などを取得できます。

Fontクラスの主なメソッド

目的	メソッド名	メソッドの概要
現在の環境で使用可能なフォントをリストアップする	`enumerateFonts`	現在のユーザ環境でSWFが使用可能なフォントを調べることができます。
そのフォントに指定の文字が含まれているか判定する	`hasGlyphs`	そのフォントで文字（列）が表示可能かを調べることができます。

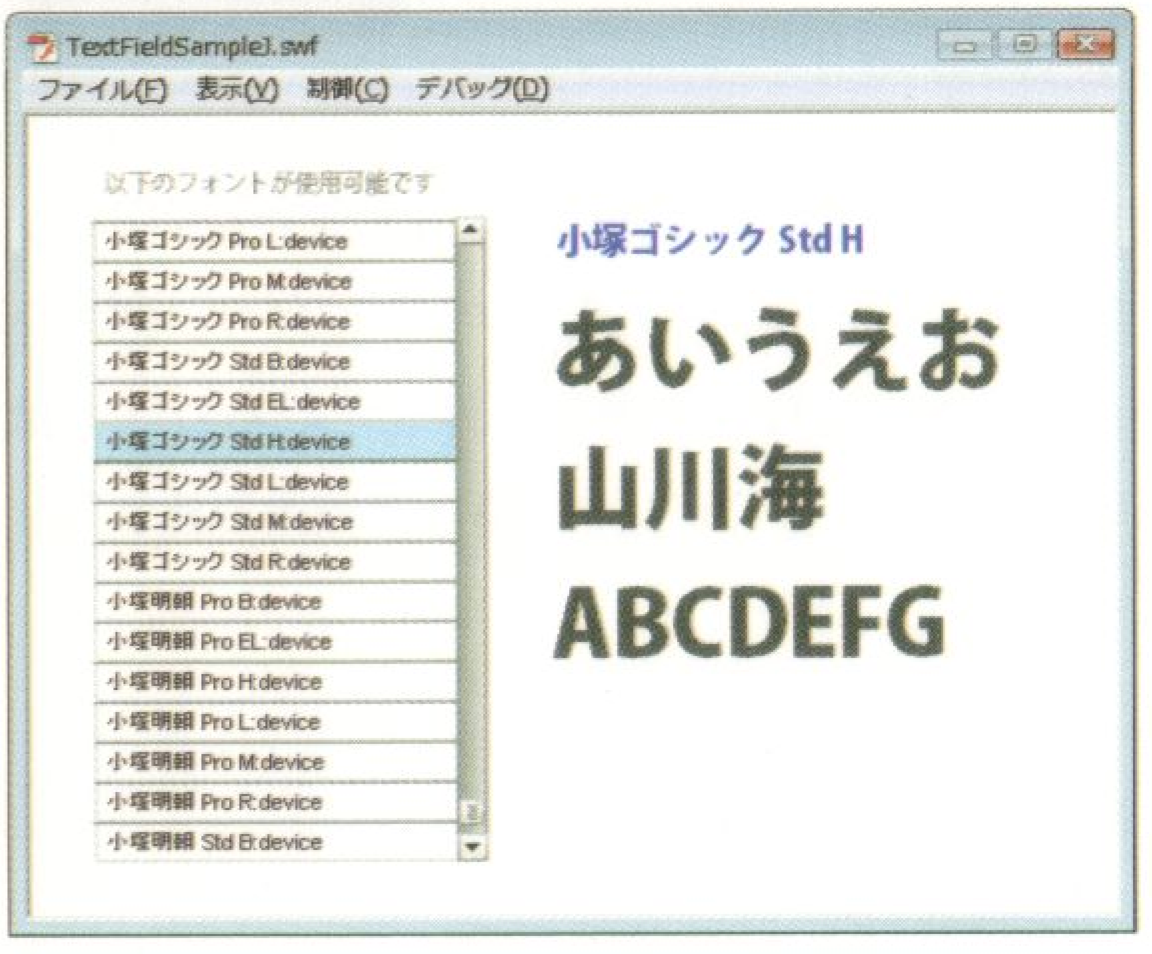
Fontクラスの主なプロパティ

目的	プロパティ名	プロパティの概要
フォント名を取得する	`fontName:String`	埋め込み、またはデバイスのフォント名を取得します。
フォントが埋め込みかデバイスフォントかを調べる	`fontType:String`	フォントが埋め込みか、デバイスフォントかを調べることができます。 **FontType.EMBEDDED**：埋め込みフォント **FontType.DEVICE**：デバイスフォント **FontType.EMBEDDED_CFF**：埋め込みのCFFフォント

Fontクラスのフォント情報を取得するためのメソッド詳細

メソッド	メソッドの詳細
`enumerateFonts (enumerateDeviceFonts:Boolean = false):Array`	使用可能なフォントのリストを配列で返します。パラメータenumerateDeviceFontsは次のように使用できます。 **true**：デバイスフォントと埋め込みフォントのリストを作成します。 **false**（デフォルト）：埋め込みフォントのリストのみを作成します。

SAMPLE



SAMPLE CODE

ユーザ環境で使用できるフォントを調べる

TextFieldSampleJ.fla

```
import flash.text.Font;
var obj:Object;
var tf:TextFormat;
//埋め込み、デバイスすべてのフォントを取得
var fontList:Array = Font.enumerateFonts(true);
//フォント名のアルファベット順で配列をソートします
fontList.sortOn("fontName", Array.CASEINSENSITIVE);
//フォントリストを作成します
for ( var i = 0 ; i < fontList.length ; i++ ) {
  obj = new Object();
  obj.label = Font(fontList[i]).fontName + ":" + Font(fontList[i]).
  fontType;
  obj.data = Font(fontList[i]).fontName;
  fontNames_list.addItem( obj );
}
fontNames_list.addEventListener( Event.CHANGE, applyFont );
//リストが選択されたときのイベントハンドラ
function applyFont(event:Event):void {
  //選択されたフォントをダイナミックテキストに適用します
  var fontName:String = List(event.target).selectedItem.data;
  tf = new TextFormat( fontName );
  sample_txt.setTextFormat( tf );
  fontName_txt.text = fontName;
  fontName_txt.setTextFormat( tf );
}
```

①ユーザー環境のすべてのフォントリストを取得

②Listコンポーネントにフォント名を表示

③選択されたフォントをダイナミックテキストに適用

5-3-10 IMEの有効無効を調べてオン/オフ、変換モードを制御する

FlashでIMEの情報を取得し、オン/オフを制御するには、CapabilitiesクラスとIMEクラスを使用します。IMEのオン/オフはWindows、Mac OSX、Linuxそれぞれでサポートされます。「かな」、「全角カナ」、「半角カナ」などの変換モードを示す定数はIMEConversionModeクラスで提供されます。

CapabilitiesクラスのIMEに関するプロパティ

目的	プロパティ	メソッドの概要
現在のシステムにIMEがインストールされているか調べる	hasIME	IMEがインストールされている場合、このプロパティはtrueになります。trueの場合、IMEクラスを使用してIMEの制御を行うことができます。

IMEクラスの主なプロパティ

目的	プロパティ名	プロパティの概要
IME変換モードを設定/取得	conversionMode:String	IME変換モードを設定するには、このプロパティにIMEConversionModeクラスの定数値をセットします。このプロパティを参照することで、現在のIME変換モードを取得できます。
IMEのオン/オフを設定/取得	enabled:Boolean	trueにすることでIMEをオン、falseでオフにできます。

IMEConversionModeクラスのIME変換モードを表す定数

変換モード	定数
全角英数	ALPHANUMERIC_FULL
半角英数	ALPHANUMERIC_HALF
全角かな	JAPANESE_HIRAGANA
全角カナ	JAPANESE_KATAKANA_FULL
半角カナ	JAPANESE_KATAKANA_HALF
中国語	CHINESE
韓国語	KOREAN

SAMPLE



SAMPLE CODE

IMEモード機能つき簡易エディタ　　TextFieldSampleK.fla

```
//IME変換モードをセットする関数
function setIME(event:Event):void {
  if ( event.target.value == "OFF" ) {  ――①
    IME.enabled = false;  ――②
  } else {
    IME.enabled = true;
    IME.conversionMode = IMEConversionMode[ event.target.value ];  ――③
  }
  //フォーカスをTextFieldに移す
  fc.setFocus( editor_txt );  ――④
}
```

IMEの設定部分のみを抜粋しています。各ラジオボタンのvalueプロパティにIMEConversionModeクラスの定数名が設定されています。例えば「全角英数」のラジオボタンのvalueには"ALPHANUMERIC_FULL"がセットされています。その値を③で取り出してIME変換モードを変更しています。「オフ」のラジオボタンには"OFF"がセットされているため、①の判定により、②でIMEをオフにしています。

④のfc変数はFocusManagerクラスのインスタンスで、フォーカスをテキストフィールドに移して編集しやすくしています。

PART 6

高度なテキスト処理

6－1 Text Layout Framework（TLF）

6-1-1 FTEとTLF

Flash Professional CS5 リリースから、Text Layout Framework（TLF）という新しいライブラリが提供され、デフォルトのテキストタイプとなりました。このTLFのベースとなるのが、Flash Playerに組み込みまれたFlash Text Engine（FTE）で、Adobe InDesignエンジンを流用した本格的なテキストレイアウトエンジンです。FTEはflash.text.engineパッケージのAPIとして提供されます。FTEは開発者が独自にコンポーネントを作成するための基礎となるように設計されており、ActionScriptを駆使して高度なテキスト処理を行うコンポーネントを作成可能です。しかし、非常に低レベルなAPIであるため、新しいテキストフレームワークとしてTLFが開発されています。TLFはFTE上に構築されたActionScript3.0コンポーネントライブラリという位置づけになります。TLFは、リッチテキストレイアウト機能とテキスト属性の詳細な設定をサポートしており、クラシックテキストと呼ばれるようになった、以前のテキストエンジンよりも詳細でリッチなテキスト操作を行うことができます。TLFをステージ上のテキストフィールドで使用する場合、fl.text.TLFTextFieldクラスを使用します。TLFTextFieldはクラシックテキストであるTextFieldと同様に、テキストの表示、編集を提供するコンポーネントのTLF版です。TLFを使用するコンテンツのパブリッシュには、ActionScript3.0およびFlash Player 10以上が必要です。

■ Text Layout Frameworkで強化された機能

機能カテゴリ	追加された点
文字スタイル	行送り、合字、ハイライトカラー、下線、打ち消し線、大文字と小文字、数字スタイルなど
段落スタイル	余白の幅を指定できる複数列のサポート、最終行の位置揃えオプション、インデント、1行目インデント、段落間隔、コンテナパディング値
日本語テキスト属性の制御	縦中横、文字組み、禁則処理タイプ、行送りモデル
属性の適用	ムービークリップシンボルに配置しなくても、3D回転、カラー効果、ブレンドモードなどの属性を適用可能
テキストフロー	複数のテキストコンテナにわたるテキストフローが可能
右←→左方向のテキストサポート	右から左に表記されるテキストに、左から右に表記されるテキストのエレメントを含めることが可能

TLFTextFieldクラスの主なプロパティ

目的	プロパティ名	プロパティの概要
アンチエイリアス処理のタイプを設定する	antiAliasType:String	このテキストフィールドに使用されるアンチエイリアス処理のタイプを表します。 **flash.text.AntiAliasType.NORMAL**：通常のテキストにアンチエイリアスを適用します。 **flash.text.AntiAliasType.ADVANCED**：文字を読みやすくする高度なアンチエイリアスを適用します。
テキストフィールドの自動拡大/縮小と整列を設定する	autoSize:String	テキストフィールドの自動的な拡大/縮小および整列を制御します。 **TextFieldAutoSize.NONE**（デフォルト）：サイズ変更は行われません。 **TextFieldAutoSize.LEFT**：左揃え **TextFieldAutoSize.RIGHT**：右揃え **TextFieldAutoSize.CENTER**：中央揃え
背景塗りつぶしがあるかを指定する	background:Boolean	テキストフィールドに背景の塗りつぶしがあるかどうかを指定します。 **true**：背景の塗りつぶしがあります。 **false**（デフォルト）：背景の塗りはありません。
背景のアルファ値を指定する	backgroundAlpha:Number	テキストフィールドの背景のアルファ値を指定します。デフォルト値：1.0
背景の色を指定する	backgroundColor:uint	テキストフィールドの背景の色を指定します。 デフォルト値: 0xFFFFFF（白）
垂直/水平方向の行配置を指定する	blockProgression:Object	垂直方向または水平方向の行配置を指定します。 **flashx.textLayout.formats.BlockProgression.TB**：上から下（横書） **flashx.textLayout.formats.BlockProgression.RL**：右から左（日本語や中国語などの縦書） **flashx.textLayout.formats.FormatValue.INHERIT**：親の値を継承します デフォルト値：undefined
境界線があるかを指定する	border:Boolean	テキストフィールドに境界線があるかどうかを指定します。 **true**：境界線があります。 **false**（デフォルト）：境界線はありません。
境界線のアルファ値を指定する	borderAlpha:Number	境界線のアルファ値を指定します。 デフォルト値：1.0
境界線の色を指定する	borderColor:uint	テキストフィールドの境界線の色を指定します。 デフォルト値：0x000000（黒）
境界線の太さを指定する	borderWidth:Number	境界線の太さを1～100の範囲で指定します。 デフォルト値：1
カーソル位置のインデックスを取得する	caretIndex:int	[読み取り専用] カーソル（キャレット）位置のインデックスを表します。フォーカスがない場合、0です。

TLFTextFieldクラスの主なプロパティ(続き)

目的	プロパティ名	プロパティの概要
テキスト列の数を指定する	columnCount :Object	テキスト列の数を1～50の整数で指定します。数値を指定しないで次の値を指定できます。 **flashx.textLayout.formats.FormatValue.AUTO**：自動的に決定します。 **flashx.textLayout.formats.FormatValue.INHERIT**：親の値を継承します。
列間のふち取り領域キャップを指定する	columnGap :Object	列間のふち取り領域を0～1000のピクセル単位で指定します。数値を指定しないで次の値を指定できます。 **flashx.textLayout.formats.FormatValue.INHERIT**：親の値を継承します。
列幅を指定する	columnWidth :Object	列幅を0～8000のピクセル単位で指定します。数値を指定しないで次の値を指定できます。 **flashx.textLayout.formats.FormatValue.AUTO**：自動的に決定します。 **flashx.textLayout.formats.FormatValue.INHERIT**：親の値を継承します。
テキストフィールド内の空白(スペース、改行など)を削除する	condenseWhite :Boolean	HTMLテキストまたはTLFマークアップが含まれるテキストフィールド内の余分な空白(スペース、改行など)を削除するかどうかを指定します。 **true**：空白文字を削除します。 **false**(デフォルト値)：空白文字を削除しません。
新しく挿入するテキストのフォーマットを指定する	defaultTextFormat:flash.text:TextFormat	新しく挿入するテキスト(ユーザが入力したテキストやreplaceSelectedText()メソッドで挿入したテキストなど)に適用するフォーマットをTextFormatオブジェクトで指定します。
左右方向の読み順を設定する	direction:String	テキストブロック内にあるテキストのデフォルトの双方向埋め込みレベルを指定します。 **flashx.textLayout.formats.Direction.LTR**：左から右への向きを指定します。 **flashx.textLayout.formats.Direction.RTL**：右から左への向きを指定します。 **flashx.textLayout.formats.FormatValue.INHERIT**：親の値を継承します。
パスワード入力用にする	displayAsPassword:Boolean	パスワード入力用かを指定します。 **true**：入力文字はアスタリスク表示されます。 **false**(デフォルト値)：入力文字はそのまま表示されます。
埋め込みフォントのアウトラインを使用するかを指定する	embedFonts :Boolean	埋め込みフォントのアウトラインを使用してレンダリングするかを指定します。 **true**：埋め込みフォントのアウトラインを使用します。 **false**(デフォルト値)：埋め込みフォントのアウトラインを使用しません。

■ TLFTextFieldクラスの主なプロパティ（続き）

目的	プロパティ名	プロパティの概要
最初の行のベースライン位置を指定する	firstBaselineOffset:Object	コンテナ内の最初の行のベースライン位置を0 ～1000の数値で指定します。下記の定数も使用できます。 **flashx.textLayout.formats.BaselineOffset.AUTO**：行のアセントとコンテナの上部のインセットを揃えます。 **flashx.textLayout.formats.BaselineOffset.ASCENT**：行のアセントと等しいオフセットを指定します。 **flashx.textLayout.formats.BaselineOffset.LINE_HEIGHT**：行の高さと等しいオフセットを指定します。 **flashx.textLayout.formats.FormatValue.INHERIT**：親の値を継承します。
グリッドフィッティングのタイプを指定する	gridFitType:String	このテキストフィールドに使用されるグリッドフィッティングのタイプです。 **flash.text.GridFitType.NONE**：グリッドフィッティングなしを指定します。 **flash.text.GridFitType.PIXEL**：太い水平線と垂直線がピクセルグリッド に合わさるように指定します。 **flash.text.GridFitType.SUBPIXEL**：太い水平線と垂直線が、LCDモニターのサブピクセルグリッドに合わさるよう指定します。
内容をHTMLで表す	htmlText:String	テキストフィールドの内容をHTMLで表します。 次の基本的なHTMLタグと<textformat>タグがサポートされます。 <a> <b> <font> <img> <i> <p> <span> <u> Part5 5-3-6 の「HTMLテキストを表示する」をご覧ください。そちらで解説しているクラシックTextFieldクラスと異なるのは、TLFTextFieldクラスでは、リストタグをサポートしないことです。
文字数を取得する	length:int	［読み取り専用］タブなどの文字も含めたテキストフィールド内の文字数を表します。
入力できる最大の文字数を指定する	maxChars:int	ユーザが入力できる最大の文字数を表します。プログラムによる入力には適用されません。 デフォルト値：0（制限なしを表します）
マウスホイールによるスクロールを有効/無効にする	mouseWheelEnabled:Boolean	ユーザがテキストフィールドをクリックしてホイールを回転させると、自動的にスクロールするかどうかを示すブール値です。 デフォルト値：true（スクロールします）
複数行かを指定する	multiline:Boolean	テキストフィールドが複数行テキストフィールドであるかどうかを示します。 デフォルト値：false（単一行）
テキスト行の数	numLines:int	［読み取り専用］複数行テキストフィールド内のテキスト行の数を取得できます。

PART
1
2
3
4
5
6
7
8
9

TLFTextFieldクラスの主なプロパティ(続き)

目的	プロパティ名	プロパティの概要
コンテナとテキストの間のスペースを指定する	paddingTop :Object paddingBottom :Object paddingLeft :Object paddingRight :Object	コンテナ下部の上端/下端/左端/右端とテキストの間のスペースを0〜1000のピクセル値で指定します。数値または次の値を指定できます。 **flashx.textLayout.formats.FormatValue.INHERIT**：親の値を継承します
ユーザが入力できる文字を制限する	restrict:String	ユーザがテキストフィールドに入力できる文字のセットを指定できます。"A-Z"のようにハイフンで範囲を指定できます。この場合、A〜Zの文字のみ入力可能になります。 デフォルト値：null (制限なし)
水平/垂直スクロール位置を取得/設定する	scrollH:int scrollV:int	現在の水平スクロール位置(ピクセル数)／垂直スクロール位置(行数)を表します。値を設定することでテキストを指定の位置までスクロールさせることができます。
テキストが選択可能かを指定する	selectable :Boolean	テキストフィールドが選択可能であるかどうかを表します。 デフォルト値：true (選択可能)
選択範囲の最初/最後のインデックスを取得する	selectionBegin Index:int selectionEnd Index:int	[読み取り専用] 現在の選択範囲の最初/最後の文字を示すインデックス値です。先頭文字のインデックスは0です。 デフォルト値は：0x000000 (黒)
テキストを取得/設定する	text:String	テキストフィールド内の現在のプレーンテキストを表します。
テキストの色を取得/設定する	textColor:uint	テキストフィールドのテキストの色を16進数形式で表します。
TLFクラスによる書式を設定するためのTextFlowオブジェクトを取得/設定する	textFlow:flashx. textLayout. elements:TextFlow	TLFクラスによる書式をTLFTextFieldフィールドインスタンスに割り当てることができます。
テキストの高さ/幅を取得する	textHeight :Number textWidth :Number	[読み取り専用] テキストの高さ/幅をピクセル単位で取得できます。
TLFマークアップを取得/設定する	tlfMarkup:String	テキストのTLMマークアップを設定または取得します。
テキストフィールドのタイプを取得/設定する	type:String	テキストフィールドのタイプを表します。 **TextFieldType.DYNAMIC** (デフォルト値)：選択可能ですが、編集することはできません。 **TextFieldType.INPUT**：編集可能なTLFテキストフィールド。

TLFTextFieldクラスの主なプロパティ(続き)

目的	プロパティ名	プロパティの概要
コピー時にフォーマットも含めるかを取得/設定する	useRichTextClipboard:Boolean	テキストと一緒にテキストのtlfMarkupフォーマットもクリップボードにコピーするかどうかを指定します。 デフォルト値:false(コピーしない)
垂直方向の行揃えまたは均等配置を取得/設定する	verticalAlign:String	垂直方向の行揃えまたは均等配置を表します。 **flashx.textLayout.formats.VerticalAlign.TOP**:フレームの上端に揃えます。 **flashx.textLayout.formats.VerticalAlign.MIDDLE**:フレームの中央に揃えます。 **flashx.textLayout.formats.VerticalAlign.BOTTOM**:フレームの下端に揃えます。 **flashx.textLayout.formats.VerticalAlign.JUSTIFY**:フレーム内での垂直方向の行の均等配置を指定します。 **flashx.textLayout.formats.FormatValue.INHERIT**:親の値を継承します。
テキストを折り返すか否かを取得/設定する	wordWrap:Boolean	テキストフィールドのテキストを折り返すか否かを表します。 デフォルト値:false(折り返さない)

TLFTextFieldクラスの主なメソッド

目的	メソッド	メソッドの概要
テキストを追加する	appendText(newText:String):void	newTextで指定されたストリングをテキストフィールド内テキストの末尾に追加します。 +演算子による追加よりも高速に行います。
現在の選択文字列を置き換える	replaceSelectedText(value:String):void	現在の選択文字列をvalueの内容に置き換えます。
範囲を指定して文字列を置き換える	replaceText(beginIndex:int, endIndex:int, newText:String):void	開始位置と終了位置を指定して文字列を置き換えます。 **beginIndex**:開始位置 **endIndex**:終了位置 **newText**:この文字列に置き換えます
範囲を指定して文字列を選択する	setSelection(begIdx:int, endIdx:int):void	開始位置と終了位置を指定して文字列を選択します。 **beginIndex**:開始位置 **endIndex**:終了位置

TLFTextFieldクラスの主なイベント

タイミング	イベント名	イベントの概要
値が変更された	change (定数Event.CHANGE)	コントロールの値が変更された後に送出されます。ユーザがテキストフィールドにテキストを入力すると、キーストロークのたびにchangeイベントが送出されます。

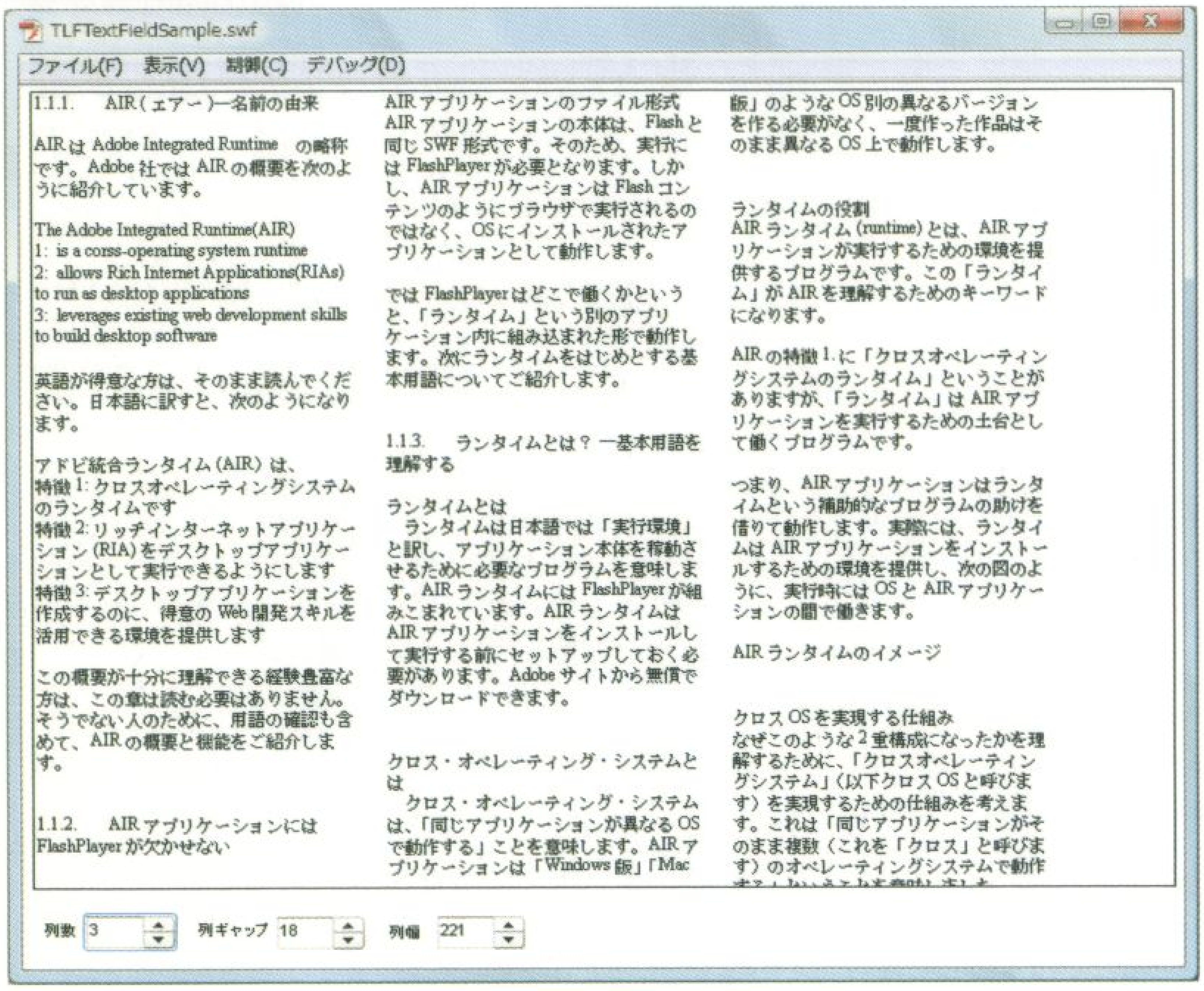
1
1

TLFTextFieldクラスの主なイベント（続き）

タイミング	イベント名	イベントの概要
ハイパーリンクをユーザがクリック	link (定数TextEvent.LINK)	HTMLが有効になっているテキストフィールドのハイパーリンクをユーザがクリックしたときに送出されます。
スクロールされた	scroll (定数Event.SCROLL)	ユーザがスクロールした後に、TextFieldオブジェクトによって送出されます。
ユーザがテキストの文字を入力した	textInput (定数TextEvent.TEXT_INPUT)	標準キーボード、IME、音声認識システム、書式やスタイルの情報を持たないプレーンテキストのペースト操作などによって、このイベントが送出されます。

PART
1
2
3
4
5
6
7
8
9

SAMPLE



SAMPLE CODE

TLFTextFieldを使う ※実行にはFlashCS5が必要

TLFTextFieldSample.fla

```
function init():void ―――――――――――――――――――――――――――――――――――― ①
{
    //TLFTextFieldのプロパティ設定
    editor_txt.multiline = true;
    editor_txt.wordWrap = true;
    //デフォルトのコラム数、ギャップ、幅を設定
    editor_txt.columnCount = 2; ―――――――――――――――――――――――――― ②
    TLF_ColumnCountNumericStepper.value = 2;
```

```
    editor_txt.columnGap = 1;────────────────────────③
    TLF_ColumnGapNumericStepper.value = 1;
    editor_txt.columnWidth = 200;
    TLF_ColumnWidthNumericStepper.value = 200;───────────④
    //イベントリスナー
    TLF_ColumnCountNumericStepper.addEventListener( Event.CHANGE,
    setTlfColumnFormat );
    TLF_ColumnGapNumericStepper.addEventListener( Event.CHANGE,
    setTlfColumnFormat );
    TLF_ColumnWidthNumericStepper.addEventListener( Event.CHANGE,
    setTlfColumnFormat );
}
//TLFによるコラム制御;
function setTlfColumnFormat(event:Event):void
{
    switch ( event.target.name )
    {
        case "TLF_ColumnCountNumericStepper" :
            editor_txt.columnCount = event.target.value;─────⑤
            break;
        case "TLF_ColumnGapNumericStepper" :
            editor_txt.columnGap = event.target.value;───────⑥
            break;
        case "TLF_ColumnWidthNumericStepper" :
            editor_txt.columnWidth = event.target.value;─────⑦
            break;
    }
}
init();
```

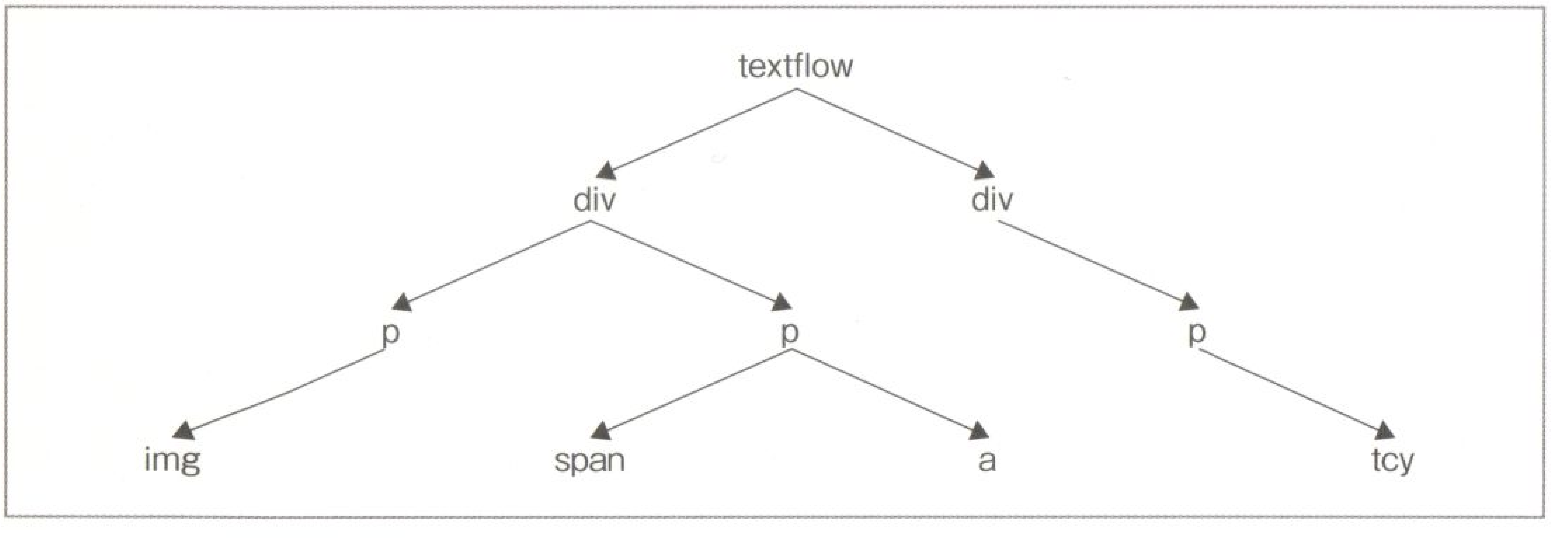
このサンプルではクラシックTextFieldにはない列情報の変化によるテキストフローの制御を確認します。①からのinit()関数でTLFTextFieldのコラムに関する初期値を設定して、NumericStepperのイベントリスナーを登録しています。②と⑤で行っているようにコラム数はColumnCount、③と⑥のようにコラムギャップはcolumnGap、④と⑦のようにコラム幅はcolumnWidthプロパティにより設定を行います。

1
1

6-1-2 TLFマークアップ言語を使う

TLFのテキストデータ構造は、TextFlowオブジェクトをルートとするflashx.textLayout.elementsパッケージクラスの階層構造で定義されます。TextFlowオブジェクトへのテキストデータの読み込みに関しては、プレーンテキストとHTMLテキスト以外に、XMLベースのマークアップ言語を使用して効率的に定義できます。その言語はTLFマークアップ言語と呼ばれ、下表「TLFマークアップ言語のエレメントとクラスの対応」にあるエレメントを使用可能です。TLFマークアップによるデータはストリング、XMLいずれでも作成して読み込むことができます。マークアップで作成したテキストを読み込む場合、TextConverterクラスのimportToFlow()メソッドにより、TextFlowオブジェクトとして取得することができます。TextConverterクラスは、TLFでこのようなインポート変換やエクスポートを行うためのクラスとして用意されています。一方、TextFlowオブジェクトを表示する場合、テキストを表示するための表示オブジェクトコンテナを管理する（表示コントローラ）IContainerControllerインターフェイスと、テキストの実際の表示レイアウトを管理するIFlowComposerインターフェイスが使用されます。これらの実装クラスとして、ContainerControllerクラスと、StandardFlowComposerクラスがあります。使用方法は、本項目のサンプルコード解説をご覧になってください。

■ TLFマークアップ言語によるテキスト構造例



■ TLFマークアップ言語のエレメントとクラスの対応

エレメント	対応クラス	機能	使用可能な子エレメント
textflow	TextFlow	ルートエレメントです。	div、p
div	DivElement	段落をグループ化します。	div、p
p	ParagraphElement	段落を表します。任意のエレメントを含めることができます。	a、tcy、span、img、tab、br

TLFマークアップ言語のエレメントとクラスの対応（続き）

エレメント	対応クラス	機能	使用可能な子エレメント
a	LinkElement	リンクを表します。	tcy、span、img、tab、br
tcy	TCYElement	縦書きの行の中で横書きのテキスト範囲を定義します。	a、span、img、tab、br
span	SpanElement	フォーマット属性セットを持つ段落内のテキスト範囲です。	—
img	InlineGraphic Element	グラフィックオブジェクト（イメージ）です。	—
tab	TabElement	タブ文字<tab/>を表します。	—
br	BreakElement	ブレーク文字です。新しい段落を作成しないで改行を作成します。	—
linkNormal Format	TextLayoutFormat	通常の状態のリンクに使用されるフォーマット属性を定義します。	—
linkActive Format	TextLayoutFormat	リンク上でマウスボタンが押されたときに、アクティブ状態のリンクに対して使用されるフォーマット属性を定義します。	—
linkHover Format	TextLayoutFormat	マウスがリンクの境界内に移動されたときに、ホバー状態のリンクに対して使用されるフォーマット属性を定義します。	—

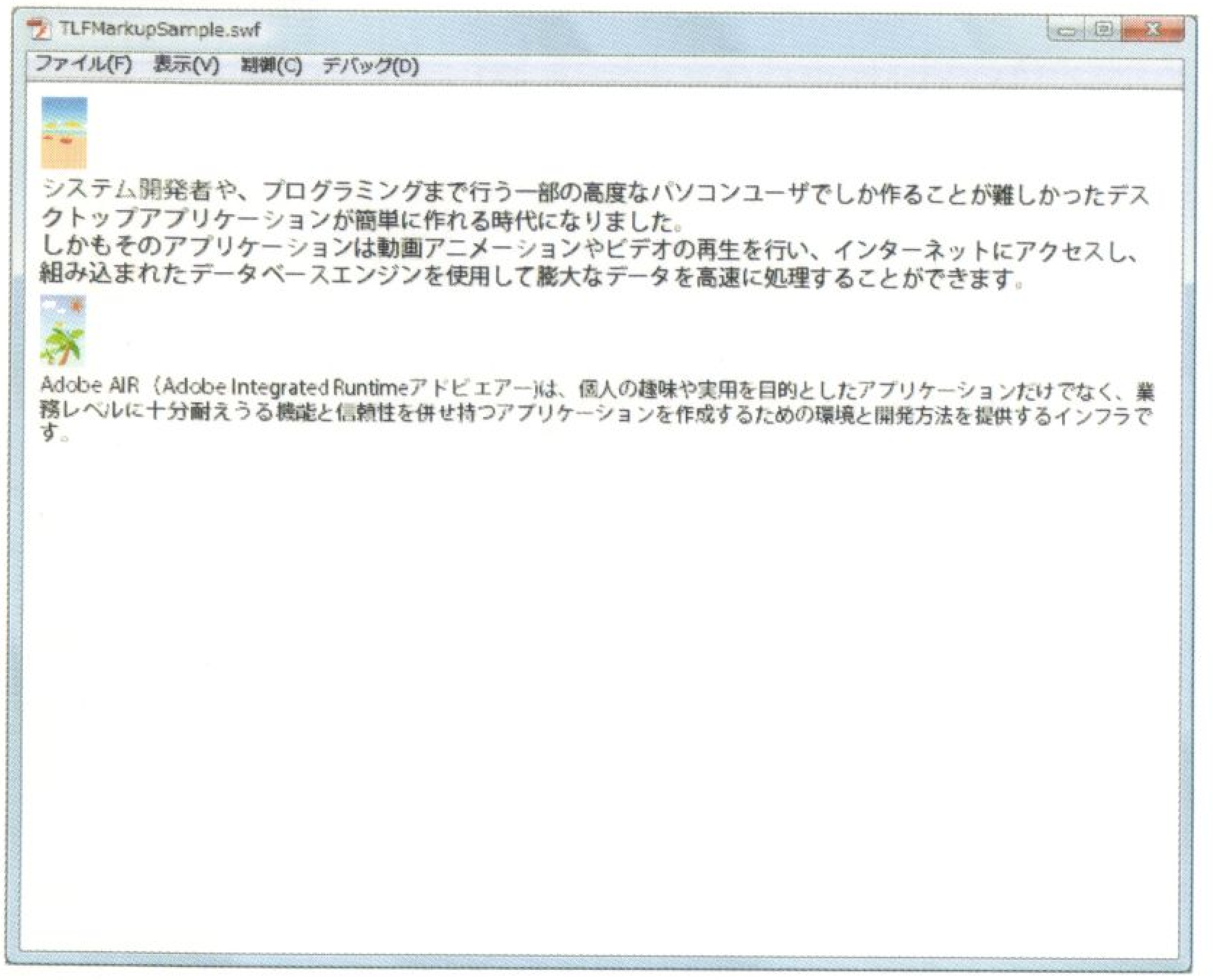
TextConverterクラスの主なメソッド

目的	メソッド	メソッドの概要
TextFlowをエクスポートする	`export(` `source:flashx.` `textLayout.elements:` `TextFlow,` `format:String,` `conversionType:String` `): Object`	［静的］sourceで与えられるTextFlowオブジェクトを指定された形式（FXG、HTML、プレーンテキスト、TLFマークアップ）にエクスポートします。 **format**： **TextConverter.TEXT_FIELD_HTML_FORMAT**： プレーンテキスト形式でエクスポートします。 **TextConverter.PLAIN_TEXT_FORMAT**： HTML形式でエクスポートします。 **TextConverter.TEXT_LAYOUT_FORMAT**： TextLayout形式でエクスポートします。 **conversionType**： ConversionType.STRING_TYPE ConversionType.XML_TYPE

TextConverterクラスの主なメソッド(続き)

目的	メソッド	メソッドの概要
TextFlowにインポートする	`importToFlow( source:Object, format:String, config:IConfiguration = null ): flashx.textLayout.elements:TextFlow`	[静的] formatで指定された形式で、sourceで与えられるソースからTextFlowを作成します。 **format**： **TextConverter.TEXT_FIELD_HTML_FORMAT**：プレーンテキスト形式でインポートします。 **TextConverter.PLAIN_TEXT_FORMAT**：HTML形式でインポートします。 **TextConverter.TEXT_LAYOUT_FORMAT**：TextLayout形式でインポートします。

SAMPLE



SAMPLE CODE

TLFマークアップによるTextFlowオブジェクトの作成と表示

※実行にはFlashCS5が必要

TLFMarkupSample.fla

```
import flash.display.Sprite;
import flashx.textLayout.elements.TextFlow;
import flashx.textLayout.formats.TextLayoutFormat;
import flashx.textLayout.container.ContainerController;
import flashx.textLayout.compose.StandardFlowComposer;
import flashx.textLayout.conversion.TextConverter;
```

```
var flowData:TextFlow;
var flowFormat:TextLayoutFormat;

//表示コンテナオブジェクトとサイズ
var textContainer:Sprite;
var containerHeight:Number = 500;
var containerWidth:Number = 780;

//TLFマークアップによるテキストデータ ―――――――――――――――― ①
var tlfMarkup:XML =
  <TextFlow xmlns="http://ns.adobe.com/textLayout/2008">
    <img source="images/summerImage2.GIF" width="30" height="50"/>
    <p>
      <span id="span1">システム開発者や、プログラミングまで行う一部の高度なパソコンユーザで
      しか作ることが難しかったデスクトップアプリケーションが簡単に作れる時代になりました。
      <br/>
      しかもそのアプリケーションは動画アニメーションやビデオの再生を行い、インターネットにアクセスし、
      組み込まれたデータベースエンジンを使用して膨大なデータを高速に処理することができます。
      </span>
    </p>
    <img source="images/summerImage3.GIF" width="30" height="50"/>
    <p>
      <span id="span2" fontSize="14">Adobe AIR(Adobe Integrated Run
      timeアドビ エアー)は、個人の趣味や実用を目的としたアプリケーションだけでなく、業務レベルに
      十分耐えうる機能と信頼性を併せ持つアプリケーションを作成するための環境と開発方法を提供するイン
      フラです。
    </span>
    </p>
  </TextFlow>;

//初期化のための関数
function init():void {
  //テキストを表示するオブジェクト
  textContainer = new Sprite();   ┐
  textContainer.x = 10;           ├――――――――――――――――――――― ②
  textContainer.y = 10;           ┘
  stage.addChild( textContainer );
  //デフォルトの書式を設定
  flowFormat = new TextLayoutFormat();
```

```
    flowFormat.color = 0x000000;
    flowFormat.fontFamily = "小塚ゴシック Pro R,_ゴシック,_明朝";
    flowFormat.fontSize = 16;
    //TextFlowオブジェクトを作成して書式を適用
    flowData = ──────────────────────────────③
    TextConverter.importToFlow(tlfMarkup,TextConverter.TEXT_LAYOUT_
    FORMAT);
    flowData.hostFormat = flowFormat;
    //TextFlowを表示するための表示コンテナコントローラを設定
    var contr:ContainerController = new ContainerController(textContain
    er,containerWidth,containerHeight);──────────④
    //テキストフローを制御するためのフローコンポーザーを設定
    flowData.flowComposer = new StandardFlowComposer();    ┐
    flowData.flowComposer.addController( contr );          ├─⑤
    flowData.flowComposer.updateAllControllers();          ┘
}

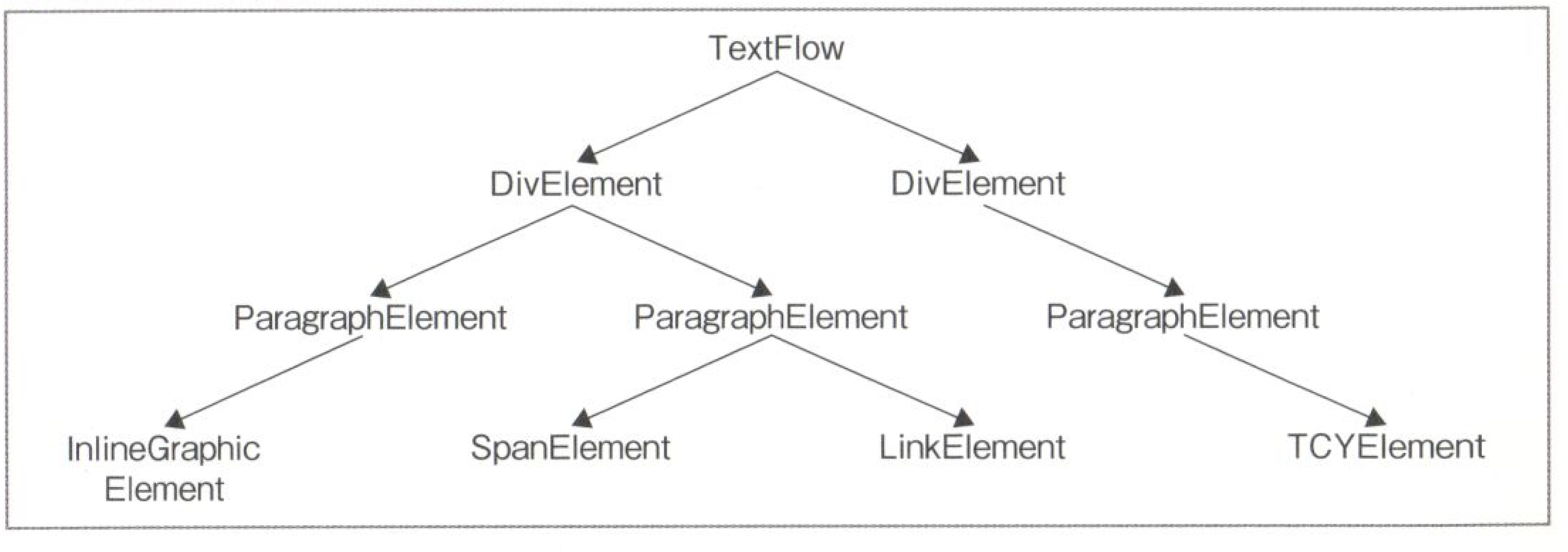
init();
```

①から定義されたXMLによるTLFマークアップデータを読み込んで、表示するサンプルです。読み込みは③で行っており、TextFlowオブジェクトを取得しています。④で表示コンテナコントローラオブジェクトを作成していますが、第1パラメータで渡しているtextContainerオブジェクトは、②で作成、表示リストに追加しているSpriteオブジェクトです。さらに、⑤で行っているように、TLFテキストのフローをコントロールするStandardFlowComposerオブジェクトを作成して、TextFlowとContainerControllerを結びつける必要があります。

6-1-3 TLFクラスを使う

TLFライブラリは、textLayout.swcファイルで提供され、コンポーネント内にはflashx.textLayoutパッケージが格納されています。このパッケージで提供されるパッケージ配下のクラスが、TLFライブラリの主要機能になります。各パッケージの概要はP.146の表「flashx.textLayout内のパッケージ」のとおりです。TLFライブラリは、大きく分けてP.146の表「TLFのクラス機能分類」の3種類のクラスに分類できます。分類は、MVCデザインパターンに類似していることから、1はモデル、2はビュー、3はコントローラとして分類されます。ただしMVCはシステム開発分野で使用されてきたデザインパターンで、必ず理解する必要はありません。TLFの場合、モデルは構造を含むテキストデータで、flow（フロー）とも呼ばれます。ビューはその名のとおり、モデルを表示オブジェクトとして表示する過程です。コントローラは、TLFの場合、ユーザの編集操作をモデルに適用し、イベント処理も提供する過程です。TLFTextFieldはクラシックテキストであるTextFieldと同様に、テキストの表示、編集を提供するTLF対応クラスです。TLFTextFieldにTLFクラスによる書式を適用する場合、textFlowプロパティで提供されるTextFlowオブジェクトを使用します。TLFでは、テキストデータの構造は、XHTMLに基づくエレメントの階層構造で定義されます。TextFlowオブジェクトはこのテキストデータ構造のルートオブジェクトとして、テキストコンテンツを管理します。TextFlowオブジェクトにスタイルを適用することにより、文書全体のスタイルとすることができます。TextFlowオブジェクトのスタイルはhostFormatプロパティに、あらかじめスタイルを設定済みのTextLayoutFormatオブジェクトを設定することで設定します。

■ TLFのテキストデータ構造例



TLFのクラス機能分類

No.	クラスの機能
1	テキスト自体の構造とスタイル、属性やテキストの読み込み、書き出しを行います
2	テキストの表示処理を行うクラス。テキストはFTEのTextLineインスタンスに変換され、Flash Playerで表示されます
3	ユーザによるテキストの編集と選択を受け付け、イベント処理も提供するクラス

flashx.textLayout内のパッケージ

パッケージ	パッケージ内クラスの概要	機能分類
compose	フローコンポーザーを定義および実装するインターフェイスとクラスが含まれます。フローコンポーザーは、TextLineインスタンスへのTextFlowの変換および1つ以上のコンテナへのTextLineインスタンスの配置を管理するオブジェクトです。	2
container	コンテナに表示されるテキストを管理するためのインターフェイスとクラスを提供します。	2
conversion	TextFlowオブジェクトに対するテキストのインポート操作またはエクスポート操作に使用するクラスを提供します。	1
edit	テキストをText Layout Frameworkで編集するためのクラスを提供します。	3
elements	TLFのテキストデータ構造はこのパッケージが提供するクラスとインターフェイスで定義されます。テキスト構造は階層ツリー形式で表現され、ツリー内の各ノードはこのパッケージ内クラスのオブジェクトになります。上図「TLFのテキストデータ構造例」に示した各エレメントはこのパッケージのクラスになります。	1
events	TLFで使用されるイベントクラスを提供します。	3
factory	読み取り専用テキストをTLFで構成、表示するためのクラスを提供します。	2
formats	テキストのフォーマットをTLFで表現するためのクラスを提供します。例えば、TextLayoutFormatクラスは、次のタイプのプロパティのフォーマットプロパティを定義します。 ・フォントファミリ、サイズ、カラー、スタイル、カーニング、トラッキング、大文字小文字などのフォントプロパティ ・行送り、均等配置の調整、テキストの行揃え、インデントなどの段落レベルのプロパティ ・複数の列、折り返し、行揃え、パディング、ベースライン位置合わせなどのコンテナレベルのプロパティ	1
operations	テキストに適用する操作をText Layout Frameworkで編集するためのクラスが含まれます。例えばEditManagerクラスは取り消しとやり直しを複数の段階で管理します。	3
utils	Text Layout Framework用のユーティリティクラスが含まれます。	-

※機能分類：番号は表「TLFのクラス機能分類」の番号で、パッケージの機能分類を示します（「-」は分類なしを示します。）

TextFlowクラスの主なプロパティ

目的	プロパティ名	プロパティの概要
Configurationオブジェクトを取得する	configuration:IConfiguration	[読み取り専用] TextFlowオブジェクトのConfigurationオブジェクトを取得できます。このプロパティを指定しない場合、defaultConfigurationが持つプロパティが使用されます。
すべての新規TextFlowオブジェクトに対して使用されるデフォルトの構成を設定する	defaultConfiguration:Configuration	[静的] すべての新規TextFlowオブジェクトに対して使用されるデフォルトの構成です。構成が明示的に指定されていない場合に使用されます。
FlowComposerオブジェクトを取得する	flowComposer:IFlowComposer	TextFlowを表示させるために使用できるIFlowComposerを実装するインスタンスにアクセスできます。このインスタンスにより、テキストを1つ以上のコンテナに関連づけ、テキストをコンテナ内で表示可能にするIFlowComposerインターフェイスで定義されたメソッドを呼び出すことができます。
TextFlowオブジェクトの世代番号を取得する	generation:uint	[読み取り専用] TextFlowオブジェクトの内容が変更されるたびに増加する世代番号を取得できます。
TextLayoutFormatオブジェクトを取得/設定する	hostFormat:flashx.textLayout.formats:ITextLayoutFormat	このTextFlowオブジェクトのフォーマットを定義するTextLayoutFormatオブジェクトを取得/設定できます。
InteractionManagerオブジェクトを取得/設定する	interactionManager:ISelectionManager	このTextFlowオブジェクトのテキストの選択と編集を制御するInteractionManagerを取得/設定できます。

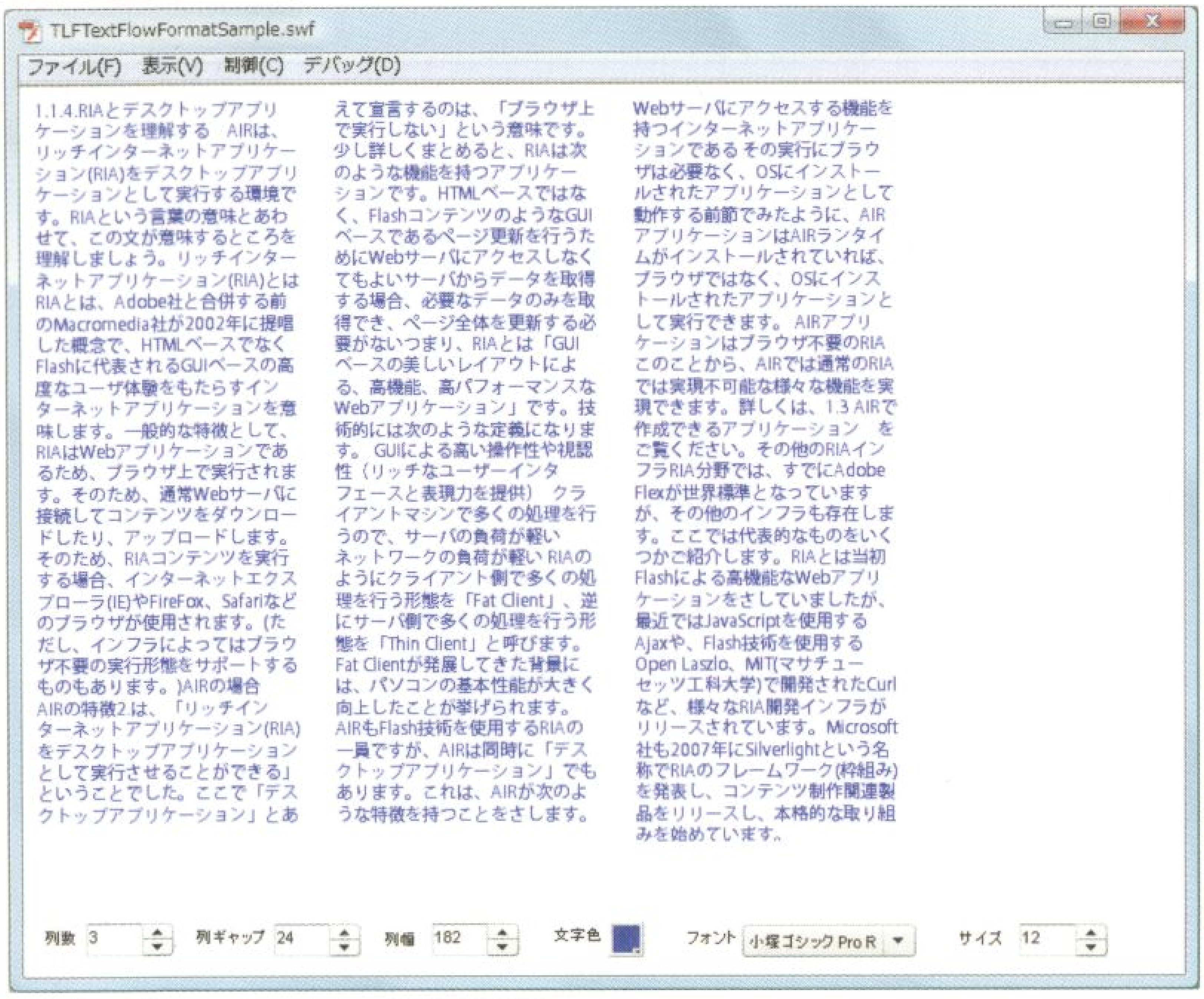
TextFlowクラスの主なメソッド

目的	メソッド	メソッドの概要
エレメントをidで取得する	getElementByID(idName:String):FlowElement	オブジェクト内でidNameに一致するidプロパティを持つエレメントを返します。
エレメントを名前で取得する	getElementsByStyleName(styleNameValue:String):Array	styleNameValueが示す値のstyleNameを持つすべてのエレメントを配列で返します。
フォーマットの再計算を強制する	invalidateAllFormats():void	スタイルを変更した後に、TextFlowのすべてのフォーマット情報を無効化し、フォーマットの再計算を行います。

TextFlowクラスの主なイベント

タイミング	イベント名	イベントの概要
リンクがクリックされた	click	リンクがクリックされると送出されます。
再構成された	compositionComplete	再構成されるたびに送出されます。
TextFlowが変更された	damage	TextFlowが変更されると送出されます。
フローオペレーションが開始する前	flowOperationBegin (定数FlowOperationEvent.FLOW_OPERATION_BEGIN)	イベントをキャンセルすると、操作がブロックされます。
フローオペレーションが終了した	flowOperationComplete (定数FlowOperationEvent.FLOW_OPERATION_COMPLETE)	フローオペレーションが完了後に送出されます。
フローオペレーションのすべての処理が終了した	flowOperationEnd (定数FlowOperationEvent.FLOW_OPERATION_END)	フローオペレーションがすべて終了して表示がスクロールされた後に送出されます。
InlineGraphicElementのサイズが変更された	inlineGraphicStatusChanged	幅または高さがautoまたはパーセントに指定されてグラフィックのロードが完了したためにInlineGraphicElementのサイズが変更された場合に送出されます。
マウス操作が行われた	mouseDown mouseUp mouseMove rollOut rollOver	各マウス操作が行われると送出されます。
テキストがコントローラコンテナ内でスクロールされた	scroll	テキストがコントローラコンテナ内でスクロールされると、TextFlowオブジェクトにより送出されます。
選択が変更された	selectionChange	テキストの選択が変更されるたびに送出されます。
コンテナの構成が変更された	updateComplete	コンテナに新規の表示オブジェクトが追加されたり、構成の結果としてコンテナが更新されるたびに、TextFlowオブジェクトにより送出されます。

SAMPLE



SAMPLE CODE

TLFTextFieldのhostFormatでTLF書式を適用する

※実行にはFlashCS5が必要

TLFTextFlowFormatSample.fla

```
import flash.events.Event;
import flashx.textLayout.formats.TextLayoutFormat;
import flashx.textLayout.elements.TextFlow;
import flashx.textLayout.container.ContainerController;
import flashx.textLayout.conversion.TextConverter;
import flashx.textLayout.compose.StandardFlowComposer;
import myPackage.TlfSamplePlainText;
//表示コンテナ
var textContainer:Sprite;
//コンテナサイズ
var containerHeight:Number = 500;
var containerWidth:Number = 780;
//テキストフローと書式のためのオブジェクト変数
var flowData:TextFlow;
var flowFormat:TextLayoutFormat;
function init():void ——————————————————— ①
{
    //テキストを表示するオブジェクト
```

```
textContainer = new Sprite();
textContainer.x = 10;
textContainer.y = 10;                          ②
stage.addChild( textContainer );
//デフォルトの書式を設定
flowFormat = new TextLayoutFormat();
flowFormat.color = 0x000000;
flowFormat.fontFamily = "小塚ゴシック Pro R,_ゴシック,_明朝";
flowFormat.fontSize = 14;                      ③
//TextFlowオブジェクトを作成して書式を適用
flowData = TextConverter.importToFlow(TlfSamplePlain
Text.plainString,TextConverter.PLAIN_TEXT_FORMAT); ④
flowData.hostFormat = flowFormat;
//デフォルトのコラム設定                          ⑤
flowData.columnCount = 2;
flowData.columnGap = 0;
flowData.columnWidth = 200;
//TextFlowを表示するためのコンポーザーを設定
var contr:ContainerController = new ContainerController
(textContainer,containerWidth,containerHeight);
flowData.flowComposer = new StandardFlowComposer();;   ⑥
flowData.flowComposer.addController( contr );
flowData.flowComposer.updateAllControllers();
//コンポーネントのコラム数、ギャップ、幅、書式値を設定;
TLF_ColumnCountNumericStepper.value = 2;
TLF_ColumnGapNumericStepper.value = 1;
TLF_ColumnWidthNumericStepper.value = 200;
colorPicker.selectedColor = 0x000000;
sizeNumericStepper.value = 14;
//イベントリスナー
TLF_ColumnCountNumericStepper.addEventListener( Event.CHANGE,
setTlfColumns );
TLF_ColumnGapNumericStepper.addEventListener( Event.CHANGE,
setTlfColumns );
TLF_ColumnWidthNumericStepper.addEventListener( Event.CHANGE,
setTlfColumns );
colorPicker.addEventListener( Event.CHANGE, setTlfFormat );
sizeNumericStepper.addEventListener( Event.CHANGE, setTlfFormat );
fontCombo.addEventListener( Event.CHANGE, setTlfFormat );
```

```
}
//TLFによるコラム制御をTextFlowオブジェクトに適用
function setTlfColumns(event:Event):void
{
  switch ( event.target.name )
  {
    case "TLF_ColumnCountNumericStepper" :
      flowData.columnCount = event.target.value;
      break;
    case "TLF_ColumnGapNumericStepper" :
      flowData.columnGap = event.target.value;
      break;
    case "TLF_ColumnWidthNumericStepper" :
      flowData.columnWidth = event.target.value;
      break;
  }
  flowData.flowComposer.updateAllControllers();
}
//TLFによるフォーマット設定;
function setTlfFormat(event:Event):void ―――――――――――――――――⑦
{
  //フォーマットを定義するTextLayoutFormatオブジェクト
  flowFormat = new TextLayoutFormat();
  flowFormat.color = colorPicker.selectedColor;
  flowFormat.fontFamily = fontCombo.selectedItem.data;
  flowFormat.fontSize = sizeNumericStepper.value;
  //TextFlowオブジェクトに適用
  flowData.hostFormat = flowFormat;
  //flowComposerのメソッドでビューを更新
  flowData.flowComposer.updateAllControllers();―――――――――――⑧
}
init();
```

①からのinit()関数で、TLFテキストのテキスト読み込み、書式設定、表示を行っています。②で表示のためのコンテナとしてSpriteオブジェクトをステージに配置しています。③でテキストデータの読み込み(④)と書式の設定を行っています。⑥で、フローデータを表示するための処理を行っています。⑤からは、コンボボックスなどの設定です。⑦からの関数で、TLFによる書式変更を行っていますが、⑧で行っているようにビューを更新する必要があります。

6-1-4 TLFエレメントのスタイル設定を行う

PART
1
2
3
4
5
6
7
8
9

TLFのテキストデータ構造は、TextFlowオブジェクトをルートとするflashx.textLayout.elementsパッケージクラスの階層構造です。それぞれのエレメントはTextFlowクラスのgetElementByID()メソッドやgetElementByStyleName()メソッドで取得できます。取得したエレメントはItextLayoutFormatインターフェイスを実装するFlowElementクラスの子孫クラスで、主にこのクラスのプロパティによってスタイルの設定を行います。

FlowElementクラスの主なプロパティ

目的	プロパティ名	プロパティの概要
ベースラインを指定する	alignmentBaseline:*	行の基準を揃えるベースラインを指定します。 **デフォルト値**：undefined
背景のアルファ値を指定する	backgroundAlpha:Number	背景のアルファ値を指定します。 **デフォルト値**：1.0
背景の色を指定する	backgroundColor:uint	背景の色を指定します。 **デフォルト値**：undefined
ベースラインを移動させる	baselineShift:*	ベースラインをdominantBaseline値から移動させる量を-1000～1000の範囲で指定します。 **デフォルト値**：undefined
垂直/水平方向の行配置を指定する	blockProgression:Object	垂直方向または水平方向の行配置を指定します。 **flashx.textLayout.formats.BlockProgression.TB**： 上から下（横書） **flashx.textLayout.formats.BlockProgression.RL**： 右から左（日本語や中国語などの縦書） **flashx.textLayout.formats.FormatValue.INHERIT**： 親の値を継承します。 **デフォルト値**：undefined
改行可能位置を制御する	breakOpportunity:*	折り返しのあるテキストを複数行に分割する際の、改行可能位置を制御します。 **flash.text.engine.BreakOpportunity.ALL**： すべての文字が、改行を行うかどうかの判断に使用されます。文字ごとに改行が行われます。 **flash.text.engine.BreakOpportunity.ANY**： オブジェクト内の文字が、改行を行うかどうかの判断に使用されます。 **flash.text.engine.BreakOpportunity.AUTO**： 文字プロパティに基づいて改行を行うかどうかを判断します。 **flash.text.engine.BreakOpportunity.NONE**： オブジェクト内の文字は、改行を行うかどうかの判断に使用されません。 **flashx.textLayout.formats.FormatValue.INHERIT**： 親の値を継承します。

FlowElementクラスの主なプロパティ(続き)

目的	プロパティ名	プロパティの概要
テキストの色を取得/設定する	color:*	RGB(赤、緑、青)値を指定する16進値を指定します。 **デフォルト値**:undefined
テキスト列の数を指定する	columnCount:*	テキスト列の数を1〜50の整数で指定します。数値を指定しないで次の値を指定できます。 **flashx.textLayout.formats.FormatValue.AUTO**: 自動的に決定します。 **flashx.textLayout.formats.FormatValue.INHERIT**: 親の値を継承します。 **デフォルト値**:undefined
列間のふち取り領域キャップを指定する	columnGap:*	列間のふち取り領域を0〜1000のピクセル単位で指定します。数値を指定しないで次の値を指定できます。 **flashx.textLayout.formats.FormatValue.INHERIT**: 親の値を継承します。 **デフォルト値**:undefined
列幅を指定する	columnWidth:*	列幅を0〜8000のピクセル単位で指定します。数値を指定しないで次の値を指定できます。 **flashx.textLayout.formats.FormatValue.AUTO**: 自動的に決定します。 **flashx.textLayout.formats.FormatValue.INHERIT**: 親の値を継承します。 **デフォルト値**:undefined
数字の大文字/小文字タイプを指定する	digitCase:*	テキストに使用される数字の大文字/小文字のタイプを表します。 **flash.text.engine.DigitCase.DEFAULT**: デフォルトの数字の大文字/小文字を使用します。 **flash.text.engine.DigitCase.LINING**: ライニング数字の大文字/小文字を指定。 **flash.text.engine.DigitCase.OLD_STYLE**: オールドスタイル数字の大文字/小文字を指定します。 lining 1234567890 oldStyle 1234567890 **flashx.textLayout.formats.FormatValue.INHERIT**: 親の値を継承します。 **デフォルト値**:undefined
数字の幅のタイプを指定する	digitWidth:*	テキストに使用される数字の幅のタイプを表します。 **flash.text.engine.DigitWidth.DEFAULT**: デフォルトの数字の幅を使用します。 **flash.text.engine.DigitWidth.PROPORTIONAL**: プロポーショナル数字の幅を指定します。 **flash.text.engine.DigitWidth.TABULAR**: 等幅数字の幅を指定します。 **flashx.textLayout.formats.FormatValue.INHERIT**: 親の値を継承します。 **デフォルト値**:undefined

PART
1
2
3
4
5
6
7
8
9

FlowElementクラスの主なプロパティ(続き)

目的	プロパティ名	プロパティの概要
左右方向の読み順を設定する	direction:*	テキストブロック内にあるテキストのデフォルトの双方向埋め込みレベルを指定します。 **flashx.textLayout.formats.Direction.LTR**：左から右への向きを指定します。 **flashx.textLayout.formats.Direction.RTL**：右から左への向きを指定します。 **flashx.textLayout.formats.FormatValue.INHERIT**：親の値を継承します。 **デフォルト値**：undefined
行エレメントの垂直座標を決めるベースラインを指定する	dominant Baseline:*	行のエレメントの垂直座標を決定するためにalignment Baselineに吸着させるエレメントのベースラインを指定します。 **flashx.textLayout.formats.FormatValue. AUTO**：親段落のlocaleプロパティに基づいて、行の基準が選択されます。 **flash.text.engine.TextBaseline.ROMAN**：欧文ベースラインを指定します。 **flash.text.engine.TextBaseline. ASCENT**：アセントのベースラインを指定します。 **flash.text.engine.TextBaseline.DESCENT**：ディセントのベースラインを指定します。 **flash.text.engine.TextBaseline.IDEOGRAPHIC_TOP**：文字の最上部のベースラインを指定します。 **flash.text.engine.TextBaseline.IDEOGRAPHIC_CENTER**：文字の中心のベースラインを指定します。 **flash.text.engine.TextBaseline.IDEOGRAPHIC_BOTTOM**：文字の最下部のベースラインを指定します。 **flashx.textLayout.formats.FormatValue.INHERIT**：親の値を継承します。 **デフォルト値**：undefined
最初の行のベースライン位置を指定する	firstBaseline Offset:*	コンテナ内の最初の行のベースライン位置を0～1000の数値で指定します。下記の定数も使用できます。 **flashx.textLayout.formats.BaselineOffset.AUTO**：行のアセントとコンテナの上部のインセットを揃えます。 **flashx.textLayout.formats.BaselineOffset.ASCENT**：行のアセントと等しいオフセットを指定します。 **flashx.textLayout.formats.BaselineOffset.LINE_HEIGHT**：行の高さと等しいオフセットを指定します。 **flashx.textLayout.formats.FormatValue.INHERIT**：親の値を継承します。 **デフォルト値**：undefined
フォント名を取得/指定する	fontFamily:*	使用するフォント名またはフォント名のカンマ区切りリストを指定できます。 **デフォルト値**：undefined

FlowElementクラスの主なプロパティ(続き)

目的	プロパティ名	プロパティの概要
フォントルックアップを取得/指定する	fontLookup:*	使用するフォント参照を表します。 **flash.text.engine.FontLookup.DEVICE**: デバイスフォントのルックアップを示します。 **flash.text.engine.FontLookup.EMBEDDED_CFF**: 埋め込みCFF(コンパクトフォントフォーマット)フォントのルックアップを示します。 **flashx.textLayout.formats.FormatValue.INHERIT**: 親の値を継承します。 **デフォルト値**:undefined
フォントサイズを取得/指定する	fontSize:*	1〜720の数値でテキストサイズを指定できます。 **flashx.textLayout.formats.FormatValue.INHERIT**: 親の値を継承します。 **デフォルト値**:undefined
フォントの斜体スタイルを取得/指定する	fontStyle:*	テキストの斜体スタイルを表します。 **flash.text.engine.FontPosture.NORMAL**: 標準のフォント傾斜を示します。 **flash.text.engine.FontPosture.ITALIC**: イタリック体の傾斜を示します。 **flashx.textLayout.formats.FormatValue.INHERIT**: 親の値を継承します。 **デフォルト値**:undefined
フォントのウェイトスタイルを取得/指定する	fontWeight:*	テキストの太さを表します。 **flash.text.engine.FontWeight.NORMAL**: 標準のフォント太さを示します。 **flash.text.engine.FontWeight.BOLD**: ボールドフォントの太さを示します。 **flashx.textLayout.formats.FormatValue.INHERIT**: 親の値を継承します。 **デフォルト値**:undefined
フォーマットを指定する	format:flashx.textLayout.formats:ITextLayoutFormat	TextLayoutFormatオブジェクトでエレメントに適用するスタイルを指定できます。
idを取得/設定する	id:String	識別名を割り当てます。TextFlowクラスのメソッドで、このidを使用してエレメントオブジェクトを取り出すことができます。
段落テキストの均等配置ルールを取得/設定する	justificationRule:*	段落のテキストの均等配置に使用するルールを表します。 **flashx.textLayout.formats.JustificationRule.EAST_ASIAN**:日中韓の均等配置ルールを示します。 **flashx.textLayout.formats.JustificationRule.SPACE**: スペースを使用して単語を分割する、ラテン語などの横書きスクリプトの均等配置を示します。 **flashx.textLayout.formats.FormatValue.AUTO**: 段落のlocaleプロパティに基づいてテキストの均等配置が行われます。 **flashx.textLayout.formats.FormatValue.INHERIT**: 親の値を継承します。 **デフォルト値**:undefined

■ FlowElementクラスの主なプロパティ(続き)

目的	プロパティ名	プロパティの概要
段落の均等配置スタイルを取得/設定する	justificationStyle:*	段落の均等配置に使用するスタイルを表します。 **flash.text.engine.JustificationStyle.PRIORITIZE_LEAST_ADJUSTMENT**：行の拡張または圧縮(必要な幅にもっとも近い結果が得られるほう)に応じて、均等配置します。 **flash.text.engine.JustificationStyle.PUSH_IN_KINSOKU**：行末の圧縮される禁則に応じて、または禁則がない場合やスペースが不足している場合の拡張に応じて、均等配置します。 **flash.text.engine.JustificationStyle.PUSH_OUT_ONLY**：行の拡張に応じて均等配置します。 **flashx.textLayout.formats.FormatValue.AUTO**：すべてのロケールでJustificationStyle.PUSH_IN_KINSOKUに解決されます。 **flashx.textLayout.formats.FormatValue.INHERIT**：親の値を継承します。 **デフォルト値**：undefined
カーニングを取得/設定する	kerning:*	カーニングがサポートされるフォントでカーニングを有効にします。 **flash.text.engine.Kerning.ON**：カーニングを有効にします。 **flash.text.engine.Kerning.OFF**：カーニングを無効にします。 **flash.text.engine.Kerning.AUTO**：アジア系文字のテキスト編集で不適切な場合を除き、カーニングを有効にします。 **flashx.textLayout.formats.FormatValue.INHERIT**：親の値を継承します。 **デフォルト値**：undefined
行送りモデルを取得/設定する	leadingModel:*	行送りの基準と行送り方向の組み合わせである、行送りモデルを指定します。 **flashx.textLayout.formats.LeadingModel.ROMAN_UP**：行送り基準がROMANで行送りの方向がUPを示します。 **flashx.textLayout.formats.LeadingModel.IDEOGRAPHIC_TOP_UP**：行送り基準がIDEOGRAPHIC_TOPで行送りの方向がUPを示します。 **flashx.textLayout.formats.LeadingModel.IDEOGRAPHIC_CENTER_UP**：行送り基準がIDEOGRAPHIC_CENTERで行送りの方向がUPを示します。• **flashx.textLayout.formats.LeadingModel.IDEOGRAPHIC_TOP_DOWN**：行送り基準がIDEOGRAPHIC_TOPで行送りの方向がDOWNを示します。 **flashx.textLayout.formats.LeadingModel.IDEOGRAPHIC_CENTER_DOWN**：行送り基準がIDEOGRAPHIC_CENTERで行送りの方向が下方向を示します。 **flashx.textLayout.formats.LeadingModel.APPROXIMATE_TEXT_FIELD**：TextFieldの行間のビヘイビアーとほぼ同じになる行送りモデルを示します。

FlowElementクラスの主なプロパティ(続き)

目的	プロパティ名	プロパティの概要
		flashx.textLayout.formats.LeadingModel.ASCENT_DESCENT_UP：行送り基準がASCENT/DESCENTで行送りの方向がUPを示します。 **flashx.textLayout.formats.LeadingModel.AUTO**：段落のlocaleプロパティに基づいて解決されます。日本語と中国語でLeadingModel.IDEOGRAPHIC_TOP_DOWN、他のすべての言語でLeadingModel.ROMAN_UP_UPとなります。 **flashx.textLayout.formats.FormatValue.INHERIT**：親の値を継承します。 **デフォルト値**：undefined
テキストの折り返しを制御する	`lineBreak:*`	テキスト折り返しを制御します。 **flashx.textLayout.formats.LineBreak.EXPLICIT**：明示的な改行文字またはラインフィード文字でのみ改行するようにします。 **flashx.textLayout.formats.LineBreak.TO_FIT**：コンテナ幅に合わせて行を折り返すようにします。 **flashx.textLayout.formats.FormatValue.INHERIT**：親の値を継承します。 **デフォルト値**：undefined
テキストの行送りを制御する	`lineHeight:*`	-720～720または-1000～1000%の範囲でテキストの行送りを制御します。 **flashx.textLayout.formats.FormatValue.INHERIT**：親の値を継承します。 **デフォルト値**：undefined
打ち消し線を描画する	`ineThrough:*`	打ち消し線の描画を指示します。 **true**：打ち消し線を表示します。設定している祖先がない場合、falseになります。 **デフォルト値**：undefined
テキストのロケールを取得/設定する	`locale:*`	テキストのロケールを表します。日本語の場合jaです。
コンテナとテキストの間のスペースを指定する	`paddingTop:*` `paddingBottom:*` `paddingLeft:*` `paddingRight:*`	コンテナの下部の上端/下端/左端/右端とテキストの間のスペースを0～1000のピクセル値で指定します。数値または次の値を指定できます。 **flashx.textLayout.formats.FormatValue.INHERIT**：親の値を継承します。 **デフォルト値**：undefined
段落の始端/終端のインデントを取得/指定する	`paragraphStartIndent:*` `paragraphEndIndent:*`	段落の始端/終端をどれだけインデントするかを0～1000のピクセル値で指定します。左から右のテキストでは右端、右から左のテキストでは左端のインデントです。 **flashx.textLayout.formats.FormatValue.INHERIT**：親の値を継承します。 **デフォルト値**：undefined

FlowElementクラスの主なプロパティ（続き）

目的	プロパティ名	プロパティの概要
段落の前/後ろに空けるスペースを取得/指定する	paragraphSpaceBefore:* paragraphSpaceAfter:*	段落の前/後に空けるスペースを0～1000のピクセル値で指定します。左から右のテキストでは右端、右から左のテキストでは左端のインデントです。 **flashx.textLayout.formats.FormatValue.INHERIT**：親の値を継承します。 **デフォルト値**：undefined
親オブジェクトを取得する	parent: FlowGroupElement	［読み取り専用］オブジェクトの親を取得できます。
レンダリングモードを取得/設定する	renderingMode:*	テキストに使用されるレンダリングモードを表します。埋め込みフォント（fontLookupプロパティがFontLookup.EMBEDDED_CFFに設定されている場合）にのみ有効です。
スタイル名を取得/設定する	styleName: String	エレメントに設定するスタイル名を表します。TextFlowクラスのメソッドで、スタイル名からエレメントを取得することができます。 **flash.text.engine.RenderingMode.NORMAL**：レンダリングモードを、Flash Player 7以前で使用されているレンダリングモードに設定します。 **flash.text.engine.RenderingMode.CFF**：CFF（コンパクトフォントフォーマット）のレンダリングモードを設定します。 **flashx.textLayout.formats.FormatValue.INHERIT**：親の値を継承します。 **デフォルト値**：undefined
タブストップを取得/設定する	tabStops:*	段落に関連づけるタブストップを指定できます。タブストップの種類は、flash.text.engine.TabAlignmentクラスで定義されている4種類です。 **CENTER**：タブで区切ったテキストの中心をタブストップに配置します。 **DECIMAL**：タブで区切ったテキストの整列トークンをタブストップに配置します。 **END**：タブで区切ったテキストの終了をタブストップに配置します。 **START**：タブで区切ったテキストの開始をタブストップに配置します。 タブストップ種類とタブ位置の情報を持つflash.text.engine.TabStopオブジェクト配列をこのtabStopsプロパティに設定できます。 start　center　decimal　end 1　あ　1.2　A 1 2　あい　23.456　AB 1 2 3　あいう　123456.7890　ABC

■ FlowElementクラスの主なプロパティ(続き)

目的	プロパティ名	プロパティの概要
段落内の行揃えを取得/設定する	textAlign:*	コンテナを基準とした段落内の行揃えを設定できます。 **flashx.textLayout.formats.TextAlign.LEFT**： 左端揃えを指定します。 **flashx.textLayout.formats.TextAlign.RIGHT**： 右端揃えを指定します。 **flashx.textLayout.formats.TextAlign.CENTER**： コンテナ内での中央揃えを指定します。 **flashx.textLayout.formats.TextAlign.JUSTIFY**： コンテナスペースを埋めるように、テキストを行内に均等配置することを指定します。 **flashx.textLayout.formats.TextAlign.START**： 始端揃えを指定します - テキストは、書き込み順と一致するように揃えられます。 **flashx.textLayout.formats.TextAlign.END**： 終端揃えを指定します。テキストは書き込み順とは反対から揃えられます。 **flashx.textLayout.formats.FormatValue.INHERIT**： 親の値を継承します。 **デフォルト値**：undefined
テキストのアルファ値を取得/設定する	textAlpha:*	テキストのアルファ(透明)値を0～1の数値で制御できます。 **flashx.textLayout.formats.FormatValue.INHERIT**： 親の値を継承します。 **デフォルト値**：undefined
テキストのアンダーライン装飾を取得/設定する	textDecoration:*	テキストのアンダーライン装飾を指定できます。 **flashx.textLayout.formats.TextDecoration.NONE**： 装飾は行いません。 **flashx.textLayout.formats.TextDecoration.UNDERLINE**：テキストにアンダーラインを引きます。 **flashx.textLayout.formats.FormatValue.INHERIT**： 親の値を継承します。 **デフォルト値**：undefined
テキストのインデントを取得/設定する	textIndent:*	段落の1行目のインデントを-1000～1000のピクセル値で指定できます。 **flashx.textLayout.formats.FormatValue.INHERIT**： 親の値を継承します。 **デフォルト値**：undefined
テキストの均等配置オプションを取得/設定する	textJustify:*	テキストの均等配置オプションを指定できます。 **flashx.textLayout.formats.TextJustify.INTER_WORD**：空白文字にスペースを追加して均等配置を実行することを指定します。 **flashx.textLayout.formats.TextJustify.DISTRIBUTE**： 空白文字と個々の文字間の両方にスペースを追加して均等配置を実行することを指定します。 **flashx.textLayout.formats.FormatValue.INHERIT**： 親の値を継承します。 **デフォルト値**：undefined

1
4

FlowElementクラスの主なプロパティ（続き）

目的	プロパティ名	プロパティの概要
テキストの長さを取得する	`textLength:int`	［読み取り専用］オブジェクトが保持するテキストの合計の長さを取得できます。
テキストの回転を取得/設定する	`textRotation:*`	テキストの回転角度を指定できます。 **flash.text.engine.TextRotation.ROTATE_0**：回転は指定しません。 **flash.text.engine.TextRotation.ROTATE_180**：180度回転を指定します。 **flash.text.engine.TextRotation.ROTATE_270**：時計回りの270度回転を指定します。 **flash.text.engine.TextRotation.ROTATE_90**：時計回りの90度回転を指定します。 **flash.text.engine.TextRotation.AUTO**：文字のUnicodeプロパティに従って、全角文字に対してのみ、反時計回りの90度回転を指定します。 **デフォルト値**：undefined
文字の左/右側のトラッキングを適用する	`trackingLeft:*` `trackingRight:*`	各文字の左/右側にトラッキング（手動カーニング）をどれだけ適用するかを、-1000～1000のピクセル単位または、-1000～1000%のfontSizeに対するパーセント値で指定できます。 **flashx.textLayout.formats.FormatValue.INHERIT**：親の値を継承します。 **デフォルト値**：undefined
文字の大文字/小文字区別のタイプを取得/設定する	`typographicCase:*`	テキストに使用される文字の大文字/小文字区別のタイプを表します。 **flashx.textLayout.formats.TLFTypographicCase.DEFAULT**：デフォルトの大文字と小文字の区別を指定します。特別な機能は適用されません。 **flashx.textLayout.formats.TLFTypographicCase.CAPS_TO_SMALL_CAPS**：出力時に大文字でスモールキャップ文字を使用するよう指定します。 **flashx.textLayout.formats.TLFTypographicCase.UPPERCASE**：出力時にすべての文字で大文字を使用するよう指定します。 **flashx.textLayout.formats.TLFTypographicCase.LOWERCASE**：出力時にすべての文字で小文字を使用するよう指定します。 **flashx.textLayout.formats.TLFTypographicCase.LOWERCASE_TO_SMALL_CAPS**：すべての小文字を大文字に変換し、変換で変更された文字にのみスモールキャップスを適用します。 **flashx.textLayout.formats.FormatValue.INHERIT**：親の値を継承します。 **デフォルト値**：undefined

FlowElementクラスの主なプロパティ(続き)

目的	プロパティ名	プロパティの概要
垂直行揃えと均等配置を取得/設定する	verticalAlign:*	垂直方向の行揃えまたは均等配置を表します。 **flashx.textLayout.formats.VerticalAlign.TOP**:フレームの上端に揃えることを指定します。 **flashx.textLayout.formats.VerticalAlign.MIDDLE**:フレームの中央に揃えることを指定します。 **flashx.textLayout.formats.VerticalAlign.BOTTOM**:フレームの下端に揃えることを指定します。 **flashx.textLayout.formats.VerticalAlign.JUSTIFY**:フレーム内での垂直方向の行の均等配置を指定します。 **flashx.textLayout.formats.FormatValue.INHERIT**:親の値を継承します。 **デフォルト値**:undefined
インポート時の空白文字の処理を指定する	whiteSpaceCollapse:*	テキストをTextFlowにインポートする際、空白を圧縮または保持する指示を行います。 **flashx.textLayout.formats.WhiteSpaceCollapse.PRESERVE**:テキストのインポート時に空白文字を維持します。 **flashx.textLayout.formats.WhiteSpaceCollapse.COLLAPSE**:テキストのインポート時に空白文字を圧縮します。 **flashx.textLayout.formats.FormatValue.INHERIT**:親の値を継承します。 **デフォルト値**:undefined

1
4

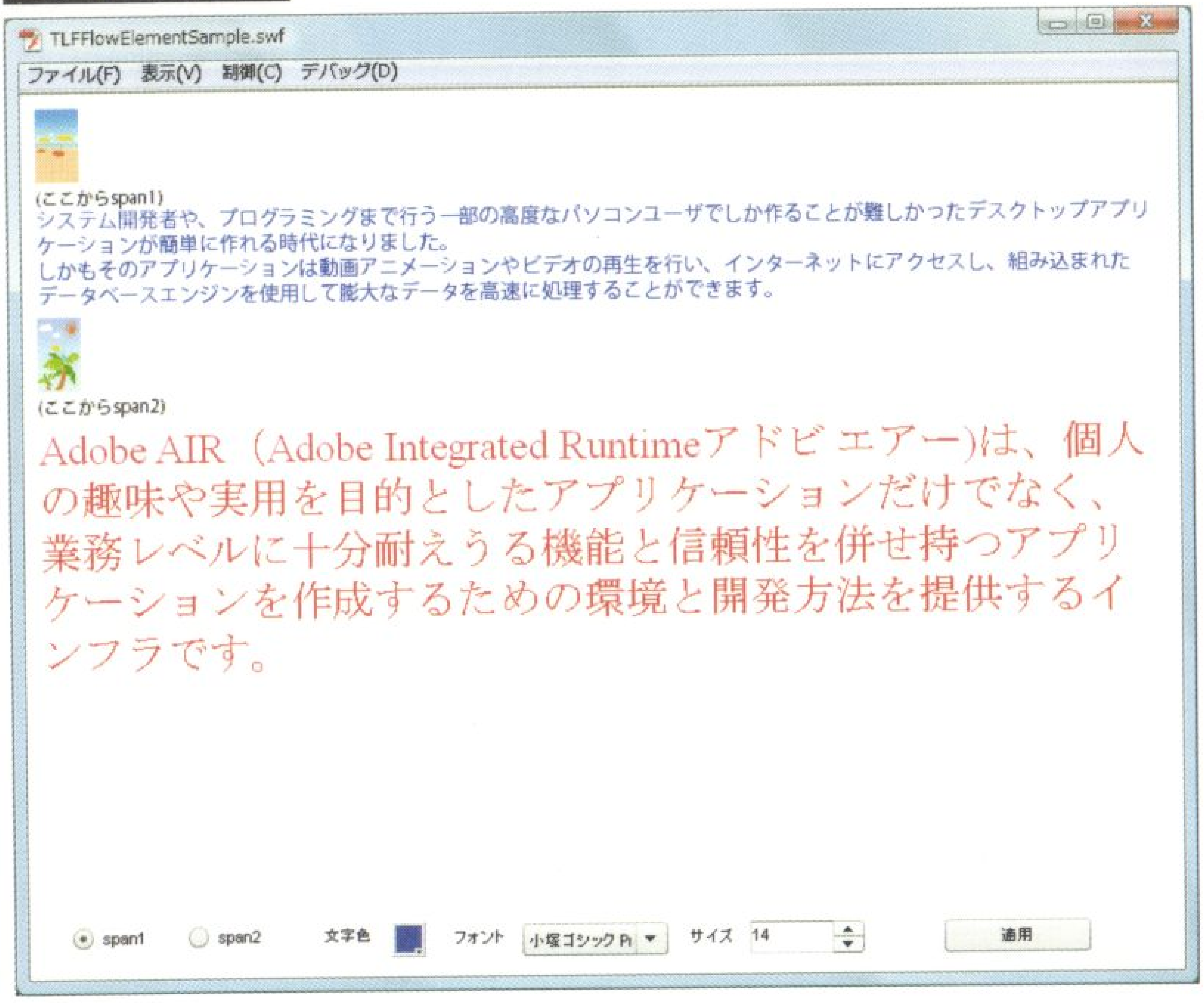
FlowElementクラスの主なメソッド

目的	メソッド	メソッドの概要
スタイルをクリアする	clearStyle(styleProp:String): void	stylePropにより指定されたスタイルをこのFlowElementオブジェクトからクリアして値をundefinedにします。
オブジェクトのコピーを作成する(子を含む)	deepCopy(relativeStart:int = 0, relativeEnd:int = -1): FlowElement	すべての子を含むFlowElementオブジェクトのコピーを作成して新しいオブジェクトとして返します。 **relativeStart**:コピー対象の最初の文字の相対テキスト位置。-1は、エレメントの終わりを表します。 **relativeEnd**:コピー対象の最後の文字の相対テキスト位置
同じタイプの次/前のFlowElementオブジェクトを取得する	getNextSibling():FlowElement getPreviousSibling(): FlowElement	テキストフロー階層内の次/前の同じタイプのFlowElementオブジェクト(兄弟)を取得します。
関連づけられているParagraphElementオブジェクトを取得する	getParagraph(): ParagraphElement	このエレメントに関連づけられているParagraphElementオブジェクトを取得します。

FlowElementクラスの主なメソッド(続き)

目的	メソッド	メソッドの概要
指定したスタイルの値を取得する	`getStyle( styleProp:String ):*`	stylePropで指定したスタイルの値を取得します。
エレメント内のテキスト範囲を取得する	`getText( relativeStart: int = 0, relativeEnd:int = -1, paragraph Separator:String = " " ): String`	フローエレメント内の指定されたテキスト範囲を取得します。 **relativeStart**:FlowElementの開始位置に対して相対的な開始位置です。 **relativeEnd**:FlowElementの開始位置に対して相対的な、テキスト範囲の終了位置です。-1は、エレメントの終わりを表します。 **paragraphSeparator**:段落間に挿入される文字を指定できます。
ルートTextFlowオブジェクトを取得する	`getTextFlow() : flashx. textLayout. elements: TextFlow`	エレメントのルートTextFlowオブジェクトを取得できます。
スタイルを適用する	`setStyle( styleProp: String, newValue:* ): void`	stylePropで指定したスタイル値を、newValueで指定する値に設定します。
オブジェクトのコピーを作成する(子を含まない)	`shallowCopy( relativeStart: int = 0, relativeEnd:int = -1 ): FlowElement`	子を含まないFlowElementオブジェクトのコピーを作成して新しいオブジェクトとして返します。 **relativeStart**:コピー対象の最初の文字の相対テキスト位置。-1は、エレメントの終わりを表します。 **relativeEnd**:コピー対象の最後の文字の相対テキスト位置
オブジェクトを分割する	`splitAtPosition( relativePosit ion:int ): FlowElement`	指定された位置でオブジェクトを分割し、その位置から始まる新しいFlowElementオブジェクトを返します。 **relativePosition**:元のオブジェクトのコンテンツを分割する位置です。最初の位置は0です。

SAMPLE



SAMPLE CODE

FlowElementによる書式変更

※実行にはFlashCS5が必要

TLFFlowelementSample.fla

```
import flash.display.Sprite;
import flashx.textLayout.elements.TextFlow;
import flashx.textLayout.formats.TextLayoutFormat;
import flashx.textLayout.container.ContainerController;
import flashx.textLayout.compose.StandardFlowComposer;
import flashx.textLayout.conversion.TextConverter;
import flashx.textLayout.elements.FlowElement;
import flashx.textLayout.elements.FlowLeafElement;
import flash.events.MouseEvent;
import flashx.textLayout.formats.TextLayoutFormat;
import fl.controls.RadioButtonGroup;
import flash.text.engine.GroupElement;
var flowData:TextFlow;
var flowFormat:TextLayoutFormat;
var flowFormat2:TextLayoutFormat;
//表示コンテナオブジェクトとサイズ
var textContainer:Sprite;
var containerHeight:Number = 500;
var containerWidth:Number = 780;
```

```
var spanGroup:RadioButtonGroup;
//TLFマークアップによるテキストデータ
var tlfMarkup:XML = ―――――――――――――――――――――――――――――――――――①
<TextFlow xmlns="http://ns.adobe.com/textLayout/2008">
<img source="images/summerImage2.GIF" width="30" height="50"/>
<div fontSize="12">(ここからspan1)</div>
<p styleName="style1">
<span id="span1">システム開発者や、プログラミングまで行う一部の高度なパソコンユーザでしか作る
ことが難しかったデスクトップアプリケーションが簡単に作れる時代になりました。
<br/>
しかもそのアプリケーションは動画アニメーションやビデオの再生を行い、インターネットにアクセスし、組み込
まれたデータベースエンジンを使用して膨大なデータを高速に処理することができます。
</span>
</p>
<img source="images/summerImage3.GIF" width="30" height="50"/>
<div fontSize="12">(ここからspan2)</div>
<p styleName="style2" fontSize="12">
<span id="span2">Adobe AIR(Adobe Integrated Runtimeアドビ エアー)は、個人の趣
味や実用を目的としたアプリケーションだけでなく、業務レベルに十分耐えうる機能と信頼性を併せ持つアプリケ
ーションを作成するための環境と開発方法を提供するインフラです。
</span>
</p>
</TextFlow>;
function init():void
{
  //テキストを表示するオブジェクト
  textContainer = new Sprite();
  textContainer.x = 10;
  textContainer.y = 10;
  stage.addChild( textContainer );
  //デフォルトの書式を設定
  flowFormat = new TextLayoutFormat();
  flowFormat.color = 0x000000;
  flowFormat.fontFamily = "小塚ゴシック Pro R,_ゴシック,_明朝";
  flowFormat.fontSize = 14;
  //TextFlowオブジェクトを作成して書式を適用
  flowData = TextConverter.importToFlow(tlfMarkup,TextConverter.TEXT_
  LAYOUT_FORMAT);
  flowData.hostFormat = flowFormat;
```

```
  //TextFlowを表示するためのコンポーザーを設定
  var contr:ContainerController = new ContainerController(textContain
  er,containerWidth,containerHeight);
  flowData.flowComposer = new StandardFlowComposer();;
  flowData.flowComposer.addController( contr );
  flowData.flowComposer.updateAllControllers();
  //ボタンのイベントハンドラ登録;
  this.set_btn.addEventListener(MouseEvent.CLICK, setElement );
  spanGroup = new RadioButtonGroup("spanGroup");
  span1_radio.group = spanGroup;
  span2_radio.group = spanGroup;
}
function setElement(event:MouseEvent):void                              ─┐
{                                                                        │
  if (spanGroup.selection == null)                                       │
  {                                                                      │
    return;                                                              │
  }                                                                      │
  //IDを指定してFlowElementオブジェクトを取得                              │
  var element:FlowElement = flowData.getElementByID(spanGroup.           ├─②
  selection.label);────────────────────────────────────────③            │
  //FlowElementの書式を設定                                               │
  element.color = colorPicker.selectedColor;                             │
  element.fontFamily = fontCombo.selectedItem.data;                      │
  element.fontSize = sizeNumericStepper.value;                           │
  //表示を更新する                                                         │
  flowData.flowComposer.updateAllControllers();──────────④              │
}                                                                       ─┘
init();
```


①からTLFマークアップデータを作成しています。マークアップデータの読み込みと表示ロジックは、6-1-3と同様です。②の関数でTextFlowオブジェクト内のFlowElementオブジェクトを取得して書式を設定しています。③でTextFlowクラスのgetElementByID()メソッドでエレメントが持つIDを指定して、FlowElementオブジェクトを取得します。取得したオブジェクトに対して行う設定はそのまま反映されますが、表示を更新するために、④の、TextFlowオブジェクトが参照するFlowComposerオブジェクトが提供するupdateAllControllers()を呼び出し、表示コントローラを更新する必要があります。

PART 7

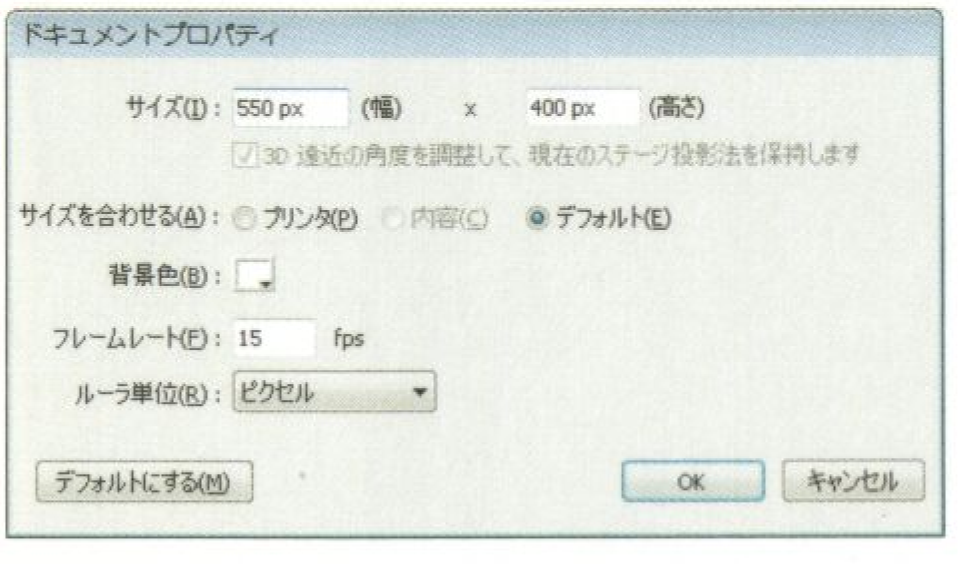
ビジュアルエレメントの操作

7-1 ステージとタイムライン

7-1-1 Stageオブジェクトを使う

ステージと呼ばれるFlashのメイン描画領域はStageクラスのオブジェクトで、アプリケーションに1つだけ存在する最上位コンテナです。Stageクラスのプロパティとメソッドにより、ActionScriptでステージサイズやフレームレートの制御などの操作ができます。Stageオブジェクトにアクセスするには、DisplayObjectオブジェクトのstageプロパティを使用します。DisplayObjectクラスはMovieClipを含むすべての表示オブジェクトのスーパークラスなので、それぞれの表示オブジェクトからstageプロパティでStageオブジェクトにアクセスできます。Flashオーサリング環境では、ドキュメントプロパティダイアログでStageオブジェクトに関するいくつかの設定を行えます。

■ ドキュメントプロパティダイアログ



■ Stageクラスの主なプロパティ概要

目的	プロパティ名	プロパティの概要
ステージの配置位置を調整する	`align:String`	Flash Player上のステージの上下左右配置を指定します。
フルスクリーンモードにする	`displayState:String`	標準/フルスクリーンモードの切り替えを行います。
フレームレートの取得/変更する	`frameRate:Number`	ステージのフレームレートを取得/設定します。
レンダリング品質を取得/変更する	`quality:String`	グラフィックとビットマップのレンダリング品質レベルを取得/変更します。次の値を使用できます。 **StageQuality.LOW**：低いレンダリング品質。グラフィックはアンチエイリアス処理されず、ビットマップはスムージングされません。 **StageQuality.MEDIUM**：普通のレンダリング品質。グラフィックは2 x 2ピクセルグリッドを使用してアンチエイリアス処理され、ビットマップのスムージングはBitmap.smoothing設定に応じて処理されます。 **StageQuality.HIGH**：高いレンダリング品質。グラフィックは4 x 4ピクセルグリッドを使用してアンチエイリアス処理され、ビットマップのスムージングはBitmap.smoothing設定に応じて処理されます。 **StageQuality.BEST**：非常に高いレンダリング品質。

Stageクラスの主なプロパティ概要（続き）

目的	プロパティ名	プロパティの概要
拡大/縮小時の動作（モード）を取得/変更する	scaleMode：String	Flash Playerとステージのサイズが異なる場合にどのようにステージサイズを調整するかを指定します。StageScaleMode
ステージのサイズを取得する	stageHeight:int stageWidth:int	現在のステージの高さ/幅を取得できます。

Stageクラスの主なイベント

タイミング	イベント名	イベントの概要
フルスクリーン/標準モード切り替え	fullScreen （定数FullScreenEvent.FULL_SCREEN）	Stageオブジェクトがフルスクリーンモードに設定されたとき、およびフルスクリーンモードから解除されたとき送出されます。
表示領域のサイズが変更	resize （定数Event.RESIZE）	Flash Playerの表示領域サイズが変更されると送出されます。つまりウィンドウのサイズが変わったときに送出されます。ただし、StageオブジェクトのscaleModeプロパティがStageScaleMode.NO_SCALEに設定しているときのみ送出されます。

SAMPLE CODE

Stageオブジェクトのイベントを確認する

StageSampleA.fla

```
//Stageインスタンスには、stageという記述でアクセスできます。
stage.scaleMode = StageScaleMode.NO_SCALE;
stage.addEventListener( Event.ACTIVATE , stageEventHandler );
stage.addEventListener( Event.RESIZE, stageEventHandler );
function stageEventHandler( event:Event ):void {
  trace( event.type + " イベントが発生しました。 " +
    "現在のステージサイズは幅=" + stage.stageWidth +
    ",高さ=" + stage.stageHeight + "です。" );
}
```

Stageクラスのactivateイベントとresizeイベントをハンドルしてステージのサイズを調べています。共通のハンドラ関数を使用していますが、イベントの種類はevent.typeプロパティで取得できます。

7-1-2 実行時にフレームレートを変更する

Stageクラスのフレームレートを変更する値を取得できます。

Stageクラスのプロパティで現在のフレームレートを取得できます。

StageクラスのframeRateプロパティで現在のフレームレートを取得できます。この値を変更すると、ムービーのフレームレートを変更することができます。フレームレートは0.01～1000の範囲で設定可能です。

Stage クラスのフレームレートに関するプロパティ

プロパティ	プロパティの詳細
frameRate: Number	使用可能なフォントのリストを配列で返します。パラメータenumerateDeviceFontsは次のように使用できます。 **true**：デバイスフォントと埋め込みフォントのリストを作成します。 **false**（デフォルト）：埋め込みフォントのリストのみを作成します。

SAMPLE



SAMPLE CODE

フレームレートを実行時に変更する

StageSampleB.fla

```
frameRate_ns.addEventListener( Event.CHANGE, changeFrameRate );
function changeFrameRate(event:Event):void {
  //ステージのフレームレートを変更
  stage.frameRate = frameRate_ns.value; ――――――――――― ①
}
```

①でStageオブジェクトのframeRateプロパティに値を設定し、フレームレートを変更しています。変更すると埋め込みビデオの再生レートを変更する効果が得られます。

7-1-3 ステージの拡大/縮小モードを変更する

StageクラスのscaleModeプロパティで、Flash Playerの表示領域（ウィンドウ）の大きさにより、アプリケーションのステージがどのように変化（拡大/縮小）するかを調整できます。scaleModeプロパティの値はStageScaleModeクラスに定数として提供されます。

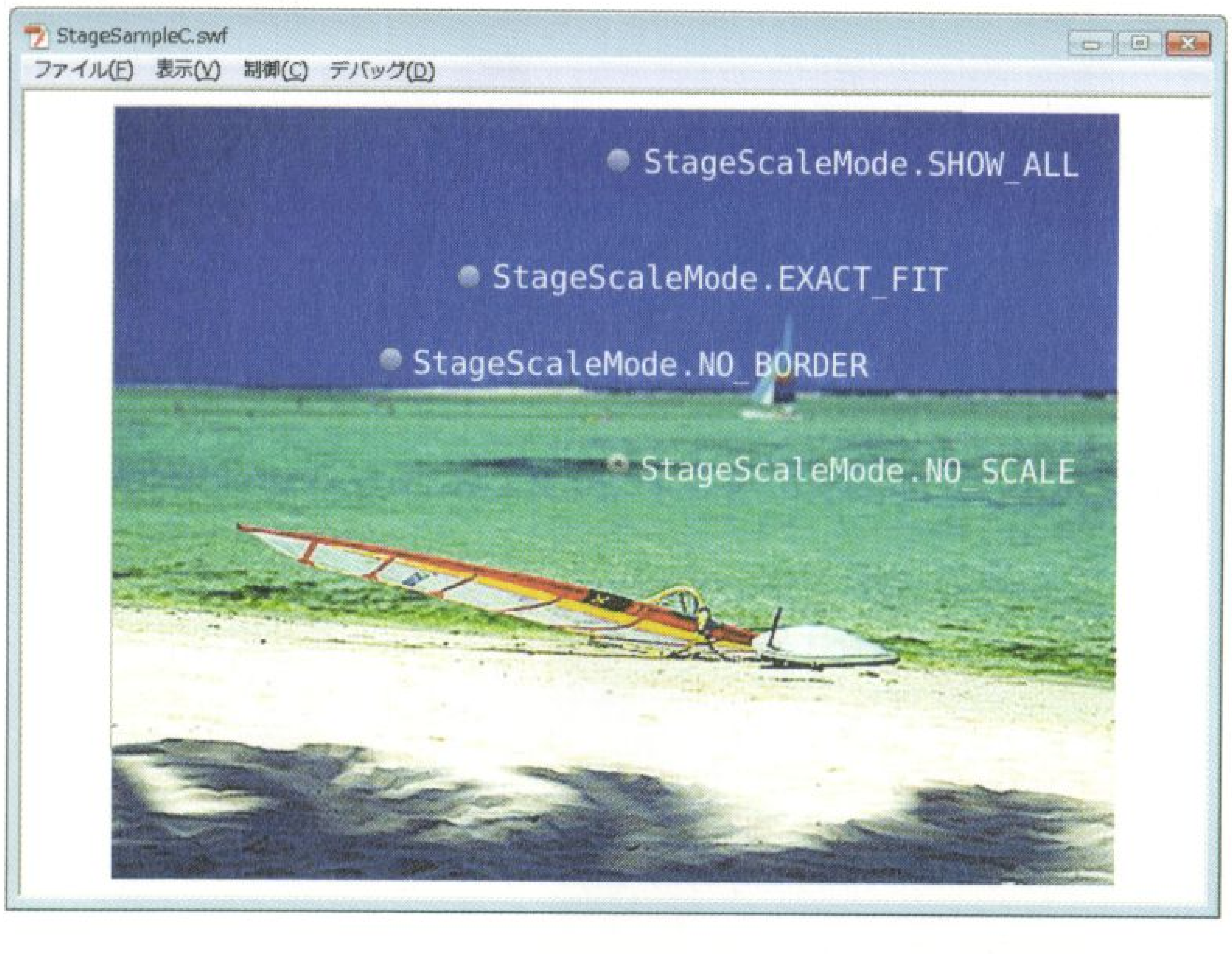
Stageクラスの拡大/縮小モードに関するプロパティ

プロパティ	プロパティの詳細
scaleMode : String	Flash Playerの表示領域に対してステージがどのように拡大/縮小するかを決定します。StageScaleModeクラスの定数プロパティ値が使用できます。

StageScaleModeクラスの定数プロパティ

定数	Stageクラスの拡大/縮小モード
NO_SCALE	アプリケーションのステージの大きさは固定になり、ウィンドウのサイズ変更に影響されません。
EXACT_FIT	ステージの縦横比は維持されず、ウィンドウのサイズにフィットします。
SHOW_ALL	ステージの縦横比が維持されたまま、できるだけウィンドウのサイズにフィットしようとします。そのため、ステージ以外の余白部分ができることがあります。
NO_BORDER	ステージの縦横比が維持されたまま、ウィンドウのサイズいっぱいに表示されます。そのため、ステージの一部がトリミングされる場合があります。

SAMPLE



ステージの拡大/縮小モードを確認するサンプルです。これがステージのオリジナルサイズ(StageScaleMode.NO_SCALE)です。ウィンドウサイズにより、ステージがFlash Player表示領域より小さいため、上下左右に余白ができています。

SAMPLE CODE

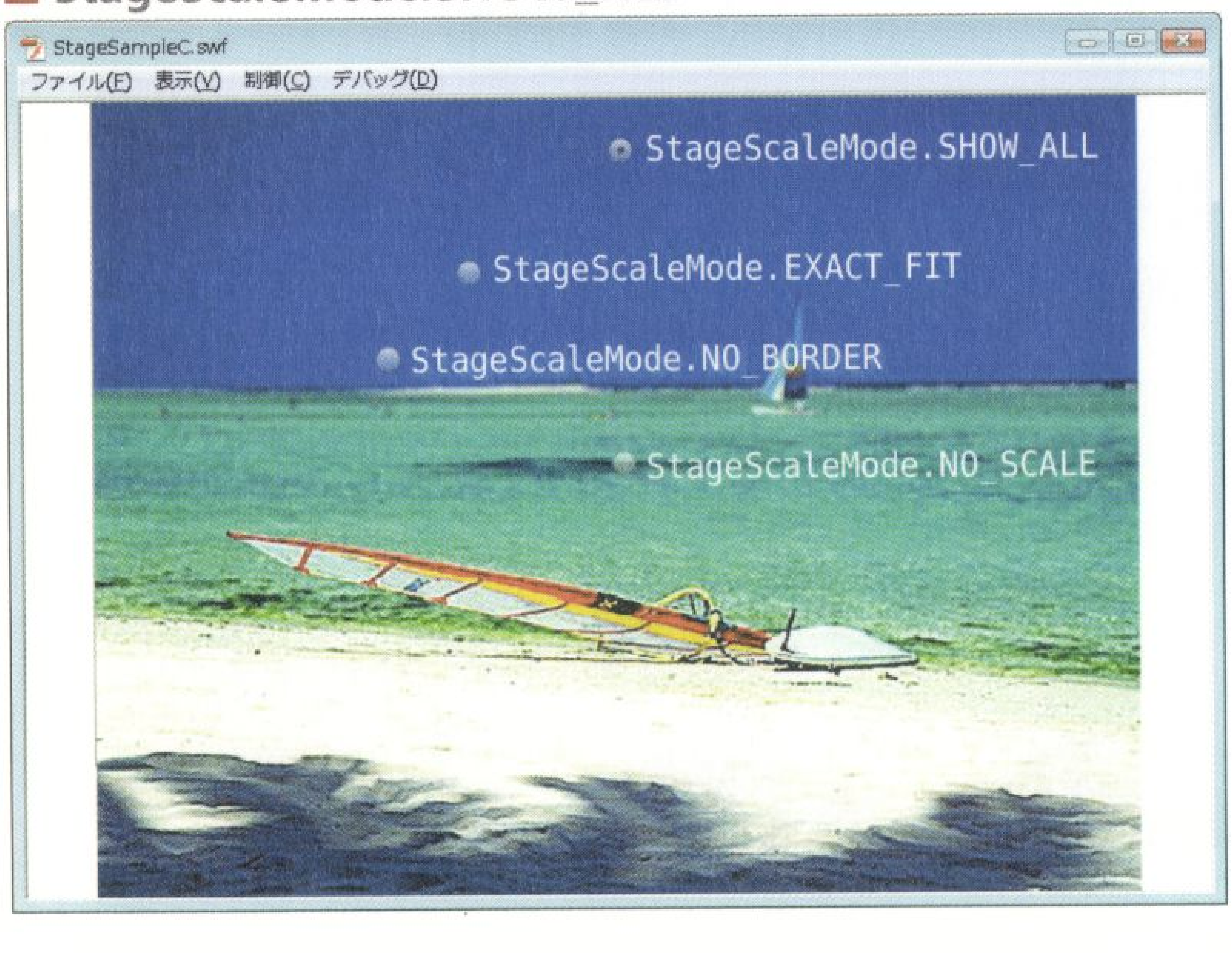
ステージの拡大/縮小モードを確認する

StageSampleC.fla

```
//ラジオボタンが押されたときのイベントハンドラ登録
this.show_all_rb.addEventListener( MouseEvent.CLICK, changeScaleMode );
this.exact_fit_rb.addEventListener( MouseEvent.CLICK, changeScaleMode
);
this.no_border_rb.addEventListener( MouseEvent.CLICK, changeScaleMode
);
this.no_scale_rb.addEventListener( MouseEvent.CLICK, changeScaleMode );
//ラジオボタンが押されたときのイベントハンドラ
function changeScaleMode(event:Event):void {
  /* scaleModeを設定
(ラジオボタンのvalueプロパティにscaleModeプロパティ用の値が設定されています)*/
  stage.scaleMode = event.target.value; ――― ①
}
```

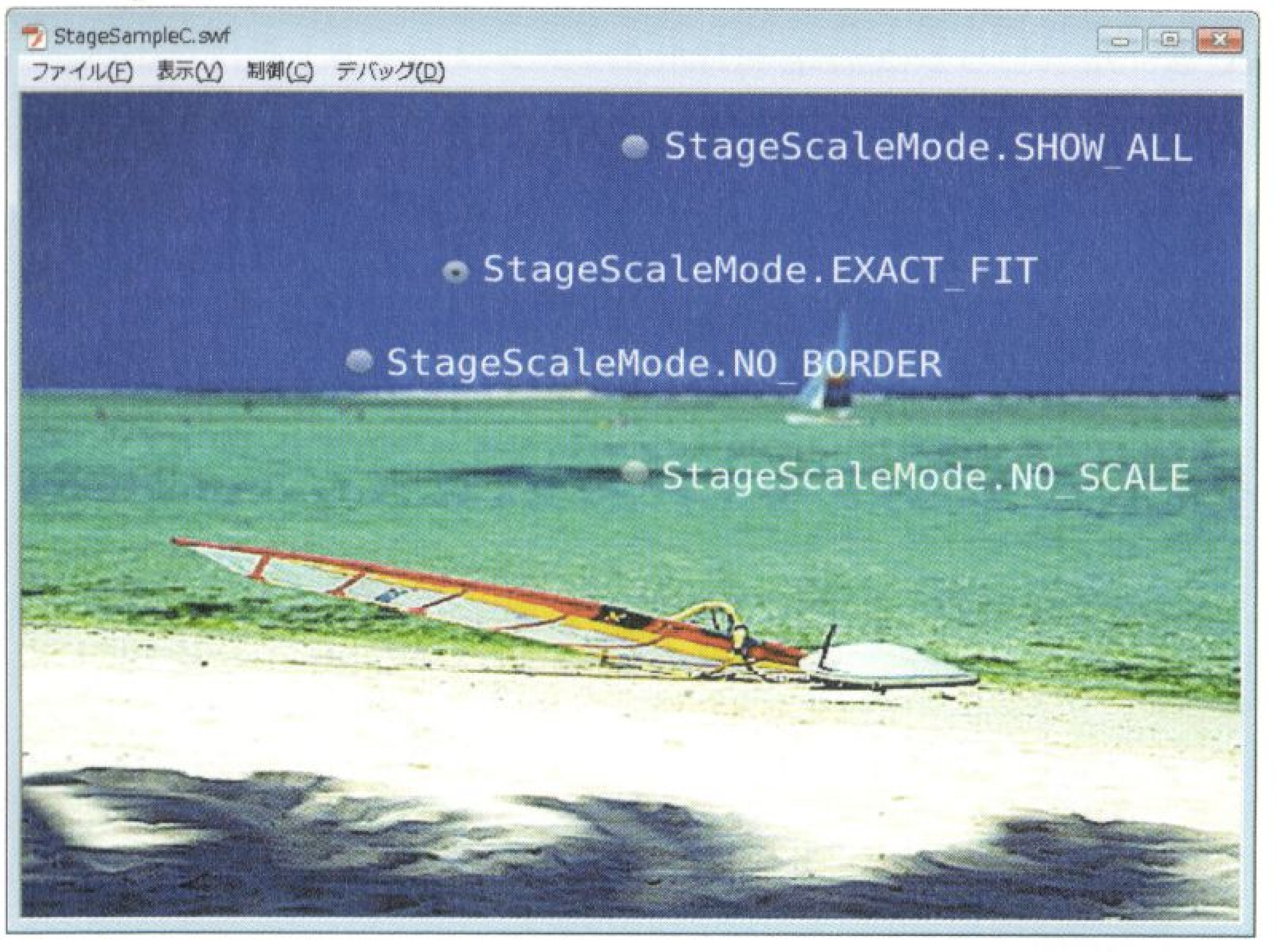
①でscaleModeを指定しています。ラジオボタンのvalueプロパティに各scaleModeの値が設定されているので、そのまま割り当てを行っています。

■ StageScaleMode.SHOW_ALL



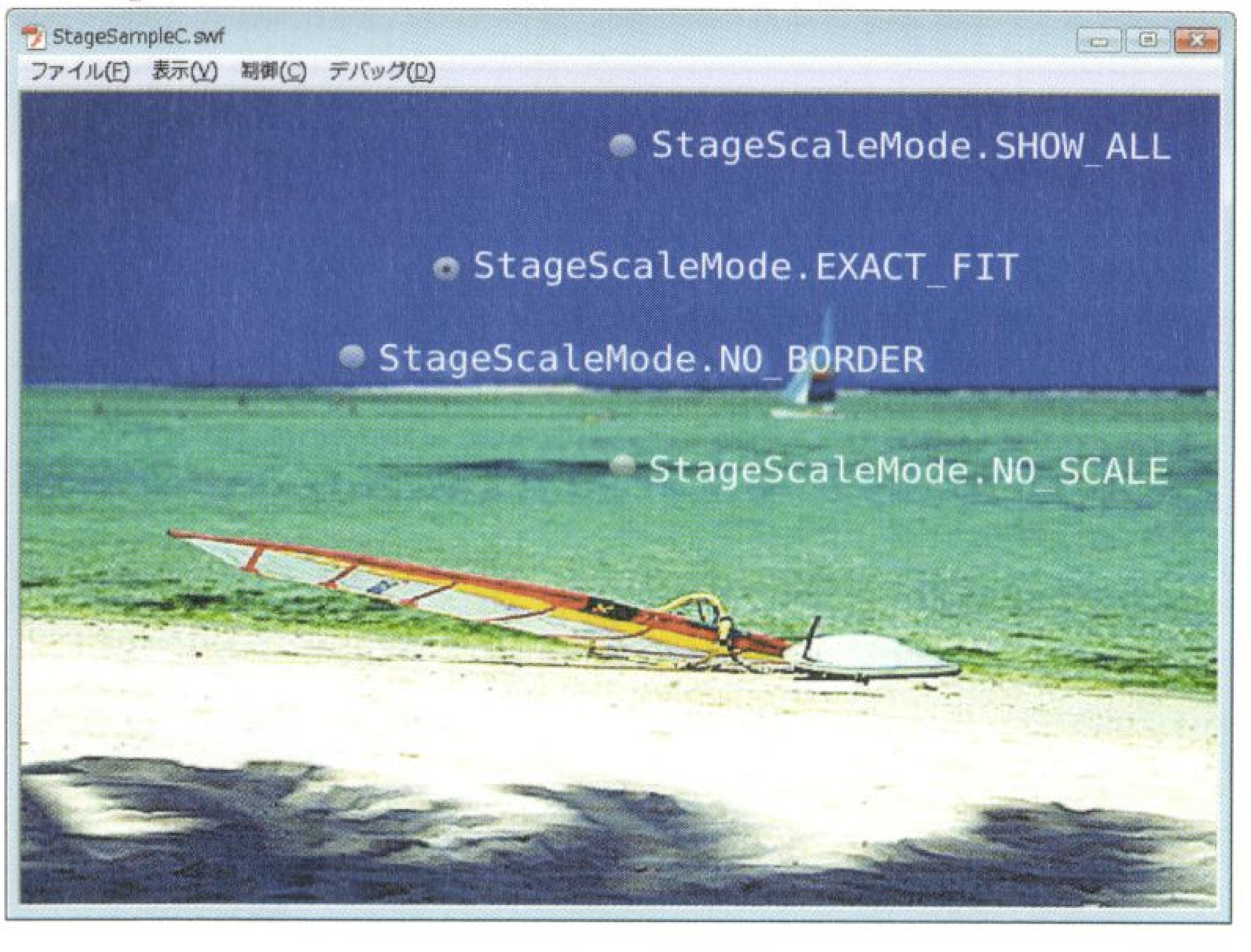
ステージの縦横比は固定でウィンドウにフィットさせます。上下にフィットして左右に余白ができています。

■ StageScaleMode.EXACT_FIT



ステージの縦横比を無視して完全にウィンドウサイズにフィットします。

■ StageScaleMode.NO_BORDER



ステージの縦横比は固定で余白がないようにウィンドウのサイズいっぱいに表示され、必要な場合トリミングされます。

1
3

7-1-4 フルスクリーンモードを使う

PART 1 2 3 4 5 6 7 8 9

Stageクラスのプロパティで、フルスクリーンモードとノーマルモードを切り替えます。Flash CS4/CS5では、パブリッシュ設定-HTMLの「テンプレート」で「Flashのみ - フルスクリーンサポート」を選択することにより、HTMLにフルスクリーンモードを許可するためのパラメータが追加され、オーサリング時にもフルスクリーンの確認が行えます。「ムービープレビュー」ではフルスクリーンモードへの切り替えはできません。また、フルスクリーンモードでは透明背景はサポートされません。

Stageクラスのフルスクリーンモードに関するプロパティ

プロパティ	プロパティの詳細
displayState: String	Flash Playerをユーザ画面全体に拡張するフルスクリーンモードに設定を行うか、標準モードで表示するかを決定します。StageDisplayStateクラスの定数プロパティ値が使用できます。

StageDisplayStateクラスの定数プロパティ

定数	Stageクラスのフルスクリーンモード
FULL_SCREEN	フルスクリーンモードを示します。
NORMAL	フルスクリーンモードでキーボード操作が有効であることを示します。
FULL_SCREEN_INTERACTIVE（AIRのみ）	標準モードを示します。

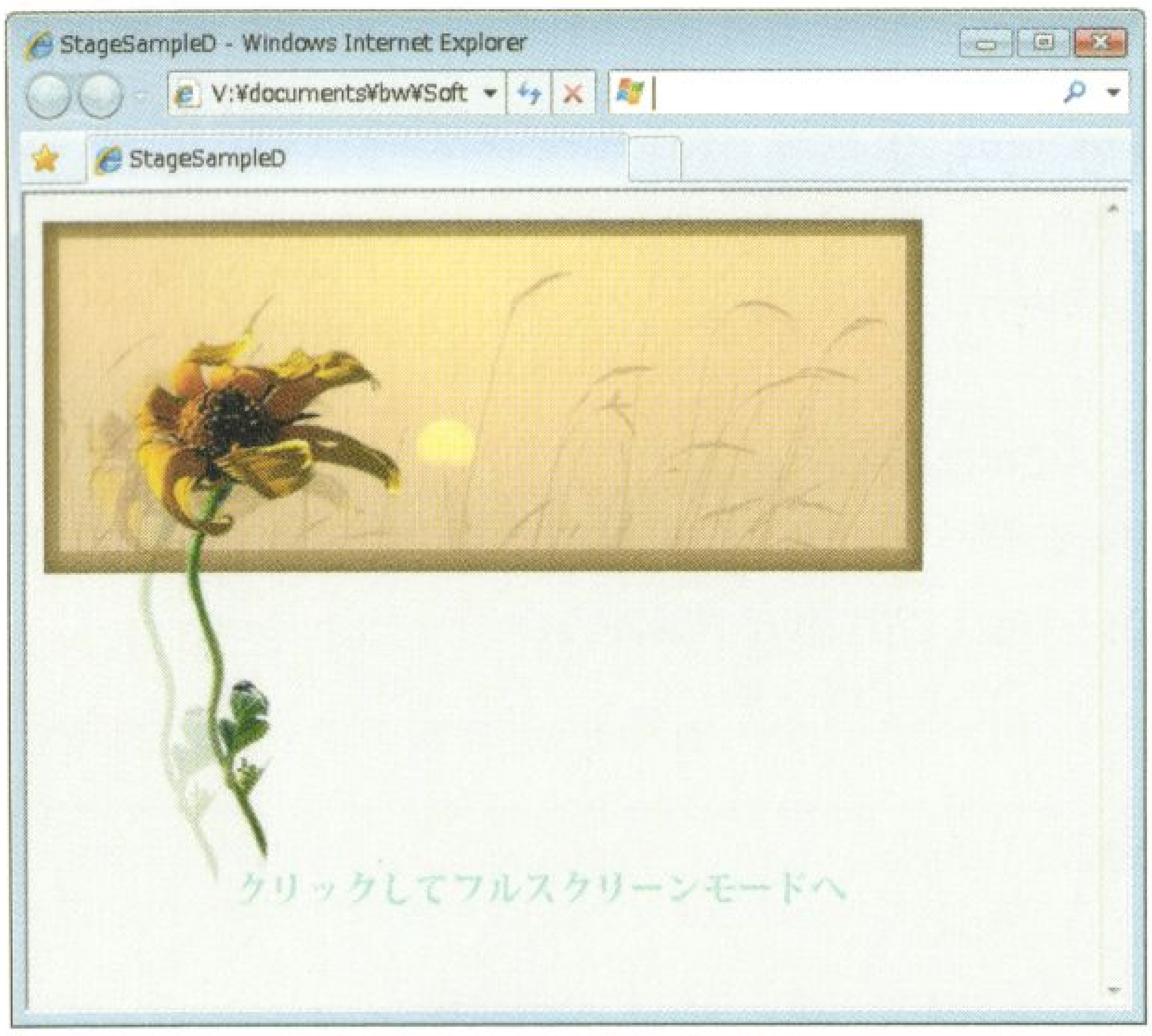
フルスクリーンモードを有効にするためのHTMLパラメータ

場所	パラメータ
<noscript>内のparamタグ	<param name="allowFullScreen" value="true" />
<noscript>内のembedタグ	<embed src="example.swf" allowFullScreen="true" ... >
AC_FL_RunContent関数のパラメータ	AC_FL_RunContent(... 'allowFullScreen', 'true', ...)

※これらのパラメータはパブリッシュ設定-HTMLの「テンプレート」で「Flashのみ - フルスクリーンサポート」を選択してパブリッシュすることで設定されます。

SAMPLE

■ 標準モード



■ フルスクリーンモード



Flash Playerが表示する、ESCキーでもフルスクリーンモードを終了できることを知らせるメッセージ

SAMPLE CODE

フルスクリーンモードと標準モードを切り替える

StageSampleD.fla

```
//テキストにイベントハンドラを登録
text_txt.addEventListener( MouseEvent.CLICK, fullScreen );
//フルスクリーンとノーマルモードを切り替える関数
function fullScreen( event:MouseEvent ):void {
  switch ( stage.displayState ) {
    case StageDisplayState.FULL_SCREEN:
      //すでにフルスクリーンのときはノーマルモードへ
      stage.displayState = StageDisplayState.NORMAL; ―――――― ①
      text_txt.text = "クリックしてフルスクリーンモードへ";
      break;
    case StageDisplayState.NORMAL:
      //ノーマルモードのときはフルスクリーンモードへ
      stage.displayState = StageDisplayState.FULL_SCREEN; ―――――― ②
      text_txt.text = "フルスクリーンモードです(クリックして戻ります)";
      break;
  }
}
```

text_txtというテキストをクリックすることでフルスクリーンモードに切り替えます。②でフルスクリーンモードへ、①で標準モードへ切り替えています。フルスクリーンモードに切り替えると、Flash Playerにより、「フルスクリーンモードを終了するにはESCキーを押します」というメッセージが表示されます。

7-2 表示オブジェクト(DisplayObject)

7-2-1 表示オブジェクト(DisplayObjectオブジェクト)を使う

ActionScript1.0、2.0では、表示エレメントはすべてMovieClipクラスに含まれていましたが、ActionScript3.0では、Flash Playerのステージに表示されるビジュアルエレメントはすべてDisplayObjectクラスを基本とするクラスとして作成されています。そのため、すべての表示エレメントには基本的な共通部分があります。表示オブジェクトの共通する使い方として、インスタンスを作成した後、DisplayObjectContainerクラスのaddChild()メソッドで表示リストに追加します。DisplayObjectContainerクラスはSpriteやStageクラスの基本クラスなので、これらのインスタンスでaddChild()メソッドを使用できます。DisplayObjectクラスは、画像、サウンド、ムービーの読み込み時に情報を提供するLoaderInforインスタンスへの参照も提供します。

DisplayObjectクラスの主なプロパティ(カラーとブレンドモード、マスク関連)概要

目的	プロパティ名	プロパティの概要
透明度(アルファ値)を取得/設定	`alpha: Number`	表示オブジェクトのアルファ透明度を表します。取得と設定が可能です。
背景オブジェクトと重なった部分のブレンドモード	`blendMode: String`	背景の表示オブジェクトと重なった部分がある場合、その部分の背景とのブレンドを設定します。 BlendModeクラスの定数が利用できます。
複雑なベクター画像の表示速度を上げる	`cacheAsBitmap: Boolean`	ベクター画像のビットマップイメージをキャッシュして表示速度を向上させます。
	`opaqueBack:ground: Object`	cacheAsBitmapとともに使用して表示速度を向上させることを目的とします。
フィルターを関連づける	`filters: Array`	flash.filtersパッケージに定義されたフィルタークラスのインスタンスにより、表示オブジェクトにフィルター処理をします。 Arrayオブジェクトを指定して、複数のフィルターを累積的に適用できます。
オブジェクトにマスクをかける	`mask: DisplayObject`	このプロパティに指定したオブジェクトにより、表示オブジェクトはマスクされます。設定したマスクを解除するには、このプロパティにnullを指定します。
オブジェクトのカラーを調整/2次元/3次元マトリックス変換を適用	`transform: Transform`	表示オブジェクトにTransformオブジェクトを割り当ててカラー調整、およびマトリックス変換を行います。Transformクラスにこれらのプロパティを設定できます。

DisplayObjectクラスの主なプロパティ(位置とサイズ変更、座標、移動、3D関連)

目的	プロパティ名	プロパティの概要
オブジェクトの高さ/幅を取得/設定する	`height`: `width`: Number	表示オブジェクトの高さ、幅をピクセル単位で表します。
マウスポインタの座標を取得する	`mouseX`: `mouseY`: Number	マウスポインタのx座標およびy座標です。単位はピクセルです。
親の表示オブジェクトコンテナ参照を取得する	`parent`: DisplayObject Container	この表示オブジェクトが格納されているDisplay ObjectContainerインスタンスを表します。
ルートの表示オブジェクトコンテナを取得する	`root`: DisplayObjectConta iner	自身を含む表示リストツリー構造のもっとも上にある表示オブジェクトを表します。表示リストに追加されていないオブジェクトでは、このプロパティはnullです。ステージを参照する場合はstageプロパティを使用します。
オブジェクトを2D回転する	`rotation`: Number	このプロパティに値を設定することで、表示オブジェクトを回転できます。単位は「度」です。
オブジェクトを3D回転する	`rotationX`: `rotationY`: `rotationZ`: Number	rotationX、rotationY、rotationZはそれぞれx軸、y軸、z軸を回転軸として表示オブジェクトを回転します。
オブジェクトの周囲を拡大/縮小用に調整する	`scale9Grid`: Rectangle	主に角が丸い四角形の四角形を拡大/縮小するときに生じる歪みを防ぐために、4辺と4つの角の8部分それぞれに拡大/縮小の規則を設けます。
オブジェクトを拡大/縮小する	`scaleX`: `scaleY`: `scaleZ`: Number	それぞれ基準点からの水平/垂直/奥行きスケールを倍率で設定します。
オブジェクトの表示部分をトリミングする	`scrollRect`: Rectangle	表示オブジェクトをトリミングします。指定するRectangleオブジェクトのx、y座標と高さ、幅がオブジェクトの「見える部分」を決める要因になります。Rectangleオブジェクトのプロパティを変化させることでトリミング領域をスクロールするなどが可能です。
ステージオブジェクトを参照する	`stage`: Stage	このプロパティでStageオブジェクトへの参照を得ることができます。
オブジェクトを表示/非表示にする	`visible`: Boolean	**true**：表示オブジェクトを表示します。 **false**：表示オブジェクトを非表示にします。
オブジェクトの位置座標を取得/設定する	`x`: `y`: `z`: Number	親となるDisplayObjectContainerを基準とした水平、垂直、奥行き座標を表します。

DisplayObjectクラスの主なプロパティ（ファイルの読み込み、ロード関連）

目的	プロパティ名	プロパティの概要
SWFファイル、画像ファイルの読み込み情報を取得する	loaderInfo :LoaderInfo	表示オブジェクトが属するSWFファイルのロード状況に関する情報を持つLoaderInfoオブジェクトにアクセスできます。

DisplayObjectクラスの主なメソッド概要

目的	メソッド名	メソッドの概要
表示オブジェクトの領域を座標で取得する	getBounds(targetCoordinateSpace: DisplayObject):Rectangle	表示オブジェクトの領域を表す矩形を取得できます。領域は境界ボックスとも呼ばれます。getBounds()メソッドは境界線を含む領域を返し、getRect()メソッドは境界線を含まない領域を返します。
	getRect(targetCoordinateSpace: DisplayObject):Rectangle	
グローバル座標 ← → ローカル座標変換する	globalToLocal(point:Point):Point	ステージ左上を基準とするグローバル座標と親DisplayObjectContainerオブジェクトを基準とするローカル座標を変換できます。 **globalToLocal()**：グローバル→ローカル変換 **localToGlobal()**：ローカル→グローバル変換 **globalToLocal3D()**：グローバル2次元ポイントをローカルの3次元座標に変換 **local3DToGlobal()**：ローカルの3次元座標をグローバル2次元ポイントに変換
	localToGlobal(point:Point):Point	
	globalToLocal3D(point:Point):Vector3D	
	local3DToGlobal(point3d:Vector3D):Point	
他の表示オブジェクトと重なっているか判定する	hitTestObject(obj:DisplayObject):Boolean	他の表示オブジェクトと重なっているかを判定します。
ある座標と接触しているか判定する	hitTestPoint(x:Number, y:Number, shapeFlag:Boolean = false):Boolean	表示オブジェクトが指定した座標と重なっているかを判定します。

PART
1
2
3
4
5
6
7
8
9

DisplayObjectクラスの主なイベント概要

タイミング	イベント名	イベントの概要
表示オブジェクトが表示リストに追加/削除された	added	**added**：表示リストに表示オブジェクトが追加されたときに発生します。 **removed**：表示リストから表示オブジェクトが削除されたときに発生します。
	removed	
	addedToStage	
	removedFromStage	
フレームレートの時間間隔で処理を行う	enterFrame	**enterFrame**：再生ヘッドが新しいフレームに入るときに送出されます。 **exitFrame**：再生ヘッドがフレームを終了するときに送出されます。 **共通事項**：再生ヘッドを移動しないか、フレームが1つの場合、同じフレームで繰り返し発生します。そのため、フレームレートの時間間隔で処理を行うのに適しています。
	exitFrame	

DisplayObjectContainerクラスの主なメソッド概要(引数は省略)

目的	メソッド名	メソッドの概要
表示オブジェクトを追加する	addChild	表示オブジェクトインスタンスを子として追加します。
	addChildAt	表示オブジェクトインスタンスを指定された位置に子として追加します。
子表示オブジェクトを取得する	getChildAt	インデックスを指定して子表示オブジェクトを取得します。
	getChildByname	名前を指定して一致する子表示オブジェクトを取得します。
子表示オブジェクトを削除する	removeChild	指定した表示オブジェクトを子リストから削除します。
	removeChildAt	インデックスで指定した表示オブジェクトを子リストから削除します。
子表示オブジェクトのインデックスを取得する	getChildIndex	指定した表示オブジェクトのインデックス位置(番号)を返します。
子表示オブジェクトのインデックスを変更する	setChildIndex	指定した表示オブジェクトのインデックス位置(番号)を、指定したものに変更します。

DisplayObjectContainerクラスの子オブジェクト追加/削除に関するメソッド詳細

メソッド	メソッドの詳細
addChild(child:DisplayObject):DisplayObject	childをDisplayObjectContainerオブジェクトの子として追加します。追加された表示オブジェクトは既存の子オブジェクトの上に配置され、前面に表示されます。childが他のDisplayObjectContainerオブジェクトの子であった場合、その子オブジェクトは削除されます。

DisplayObjectContainerクラスの子オブジェクト追加/削除に関するメソッド詳細(続き)

メソッド	メソッドの詳細
addChildAt(child:DisplayObjact, index:int):DisplayObject	childをDisplayObjectContainerオブジェクトの子として追加します。indexにより、位置を指定して追加できます。位置は0が最背面で1、2...の順に前面に追加されます。すでに使用されている番号を指定した場合、その番号以上の子オブジェクトのインデックスは繰り上げられます。childが他のDisplayObjectContainerオブジェクトの子であった場合、その子オブジェクトは削除されます。
removeChild(child:DisplayObject):DisplayObject	childで指定した表示オブジェクトを表示リストから削除します。削除された表示オブジェクトのparentプロパティがnullに設定されます。
removeChildAt(index:int):DisplayObject	indexで指定したインデックスを持つ表示オブジェクトを表示リストから削除します。削除された表示オブジェクトのparentプロパティがnullに設定されます。

SAMPLE





テキストをクリックすると、ロード済みのイメージを表示リストに追加します。

SAMPLE CODE

表示オブジェクトを表示リストに追加する DisplayObjectSampleA.fla

```
import flash.display.Loader;
import flash.events.Event;
import flash.events.MouseEvent;
import flash.net.URLRequest;

//ロードするファイル名
var url:String="images/Halloween.png";
var t:TextField;
var loader:Loader;
```

```
LoadImage();

//画像をロードする関数
function LoadImage() {
  loader = new Loader();
  loader.contentLoaderInfo.addEventListener(Event.COMPLETE,
  completeHandler);
  var request:URLRequest=new URLRequest(url);
  loader.load(request);
}

//ロード完了時のハンドラ関数
function completeHandler(event:Event):void {
  t = new TextField();
  t.autoSize=TextFieldAutoSize.LEFT;
  t.multiline=true;
  t.text="イメージのロードが完了しました。¥nクリックして表示リストに追加します。";
  t.type=TextFieldType.DYNAMIC;
  t.textColor=0xFFFFFF;
  t.addEventListener(MouseEvent.CLICK, addLoader );
  //テキストフィールドを表示リストに追加
  addChild( t );————————————————————①
}

//テキストクリック時のハンドラ関数
function addLoader(event:Event):void {
  //テキストフィールドを表示リストから削除
  removeChild( t );—————————————————②
  //Loaderオブジェクトを表示リストに追加
  addChild(loader);—————————————————③
}
```

表示オブジェクトのプログラミングを行う場合にもっとも基本となる、表示リストへの追加と削除を確認するサンプルです。①では、作成したテキストフィールドオブジェクトをaddChild()メソッドにより表示リストへ追加しています。そのオブジェクトは②でremoveChild()メソッドにより削除しています。読み込んだイメージファイルはLoaderオブジェクトの子として格納されているので、Loaderオブジェクトを表示リストに追加することで、画面に表示されるようになります。③でその処理を行っています。

7-2-2 表示オブジェクトの位置とサイズ変更、回転

表示オブジェクトの位置とサイズ、回転はプロパティ値を変更することにより、調整できます。位置に関するプロパティは、ステージ左上を基準とするグローバル座標と、親DisplayObjectContainerの左上を基準とするローカル座標の2種類があります。DisplayObjectクラスが持つx、y、zの位置情報はローカル座標を表します。このことにより、親コンテナを移動、回転すると子のDisplayObjectも一緒に移動、回転することになります。

DisplayObjectクラスの位置とサイズに関するプロパティ詳細

プロパティ	プロパティの詳細
height width :Number	表示オブジェクトの高さをピクセル単位で表します。これらプロパティにより、オブジェクトのサイズを変更すると、対応するscaleY（heightと対応）、scaleX（widthと対応）も変化します。
rotation :Number	2次元の回転を行います。このプロパティに値を設定することで、表示オブジェクトを回転できます。単位は「度」です。プラス値で時計回り、マイナス値を設定して反時計回りに回転できます。360を超える値の場合、360（の倍数）を引き、-360を下回る場合、足すことで角度が調整されます。
rotationX rotationY rotationZ :Number	x軸/y軸/z軸を回転軸として表示オブジェクトを回転します。 Flash Player10/AIR1.5以降のActionScript表示オブジェクトはすべてzプロパティを持ち、z軸を利用した3D座標を使用できます。z軸はオブジェクトを視点から遠ざかる位置をコントロールします。
scaleX scaleY scaleZ :Number	基準点からの水平/垂直/奥行きスケールを倍率で設定します。100%の大きさで表示するには、1.0を指定します。scaleX、scaleYを変更するとheight、widthプロパティの値も変化します。負の値を設定することにより、左右（scaleX）、上下（scaleY）の反転も行えます。
x y z :Number	x軸/y軸/z軸方向のローカル座標を表します。親のDisplayObjectContainerオブジェクトが移動や回転をする場合、ローカル座標系も一緒に移動、回転します。つまり、親が90度回転した場合、ローカル座標のxは縦方向の位置を表すことになります。

2
2

SAMPLE





SAMPLE CODE

ステージの拡大/縮小モードを確認する

DisplayObjectSampleB.fla

```
import flash.display.Loader;
import flash.events.*;
import flash.net.URLRequest;

//ロードするファイル名
var url:String="Halloween.png";
var t:TextField;
var loader:Loader;
LoadImage();

//画像をロードする関数
function LoadImage() {
  loader = new Loader();
  loader.contentLoaderInfo.addEventListener(Event.COMPLETE,
  completeHandler);
  var request:URLRequest=new URLRequest(url);
  loader.load(request);
}

//ロード完了時のハンドラ関数
function completeHandler(event:Event):void {
  trace("completeHandler: " + event);
  t = new TextField();
  t.autoSize=TextFieldAutoSize.LEFT;
  t.multiline=true;
  t.text="イメージのロードが完了しました。¥nクリックして表示リストに追加します。";
  t.type=TextFieldType.DYNAMIC;
  t.textColor=0xFFFFFF;
  t.addEventListener(MouseEvent.CLICK, addLoader );
  //テキストフィールドを表示リストに追加して位置を調整
  addChild( t ); t.x = (stage.stageWidth - t.width ) / 2; t.y = 200;
}                                                                 └①

//テキストクリック時のハンドラ関数
function addLoader(event:Event):void {
  //テキストフィールドを表示リストから削除
```

```
    removeChild( t );
    //読み込んだBitmapオブジェクトのサイズと位置を調整
    var bmp:Bitmap = loader.content as Bitmap; ——②
    bmp.x = ( stage.stageWidth - bmp.width ) / 2 + 50;
    bmp.y = ( stage.stageHeight - bmp.height ) / 2;  ——③
    bmp.width *= 1.7; bmp.scaleY = 1.7; bmp.rotationY = 45;
    //Loaderオブジェクトを表示リストに追加
    addChild(loader);
}
```

①でテキストフィールドを評価リストに追加してからx座標をステージ幅との差を計算して指定しています。この式により、ステージの横中央に配置します。y座標は固定の200です。LoaderオブジェクトのcontentプロパティでロードしたファイルをSpace参照できます。

今回はイメージファイルをロードしたので、contentはBitmapオブジェクトになります。②でBitmapオブジェクトを取得してbmp変数に格納しています。③でビットマップの座標、サイズ、3Dローテーションを調整しています。

7-2-3 表示オブジェクトの透明度、マスク、ブレンドモード、描画を調整する

表示オブジェクトの透明度、マスク、ブレンドモード、ベクター描画の効率化などはDisplayObjectのプロパティで設定することができます。ブレンドモードはBlendModeクラスに定数が定義されており、それを指定することで適用できます。

DisplayObjectクラスの透明度、マスク、ブレンドモード、描画に関するプロパティ詳細

プロパティ	プロパティの詳細
alpha :Number	表示オブジェクトのアルファ透明度をNumber型の数値で設定します。0.0～1.0の範囲が指定できます。0に近くなるほど透明度が高くなり、0.0で完全に透明になります。デフォルトは1.0で、これは透明度がないことを表します。
blendMode :String	設定を行った表示オブジェクトのピクセルを構成する赤、緑、青の各要素について、背景と指定されたブレンドを行います。設定は決められたブレンド名をStringで指定しますが、BlendModeクラスに定数が用意されています。
mask :DisplayObject	このプロパティに設定された表示オブジェクトでマスク処理を行います。マスクを解除するには、プロパティをnullにします。
cacheAsBitmap :Boolean	**true**：ベクター画像のビットマップイメージをキャッシュして表示速度を向上させることを目的とします。画像の条件により、効果が変わります。 **false**：ビットマップイメージは作成されません。
opaqueBack ground:Object	オブジェクトの背景のうち、指定した色についてアルファチャンネル(透明色)の描画を行わずにパフォーマンスを向上させます。cacheAsBitmapとともに使用して表示速度を向上させることを目的とします。

BlendModeクラスの定数とブレンドの内容

目的	定数名	ブレンドの内容
前景のピクセルを優先させる(デフォルト)	BlendMode.NORMAL	デフォルトのブレンド。表示オブジェクトのピクセル値は、背景のピクセル値より優先され、背景の前に表示されます。透明な部分では、背景が表示されます。
2つのイメージ間での色を薄くするディゾルブをアニメーション処理する	BlendMode.ADD	表示オブジェクトの要素カラーの値を背景色に加算し、その際に上限0xFFを適用します。
前景のアルファを背景に適用する	BlendMode.ALPHA	表示オブジェクトの各ピクセルのアルファ値を背景に適用します。親表示オブジェクトのblendModeがBlendMode.LAYERに設定されている必要があります。
重ね合わせ表示	BlendMode.DARKEN	表示オブジェクトの要素カラーと背景色のうち暗いほう(値が小さいほう)の色を選択します。

BlendModeクラスの定数とブレンドの内容（続き）

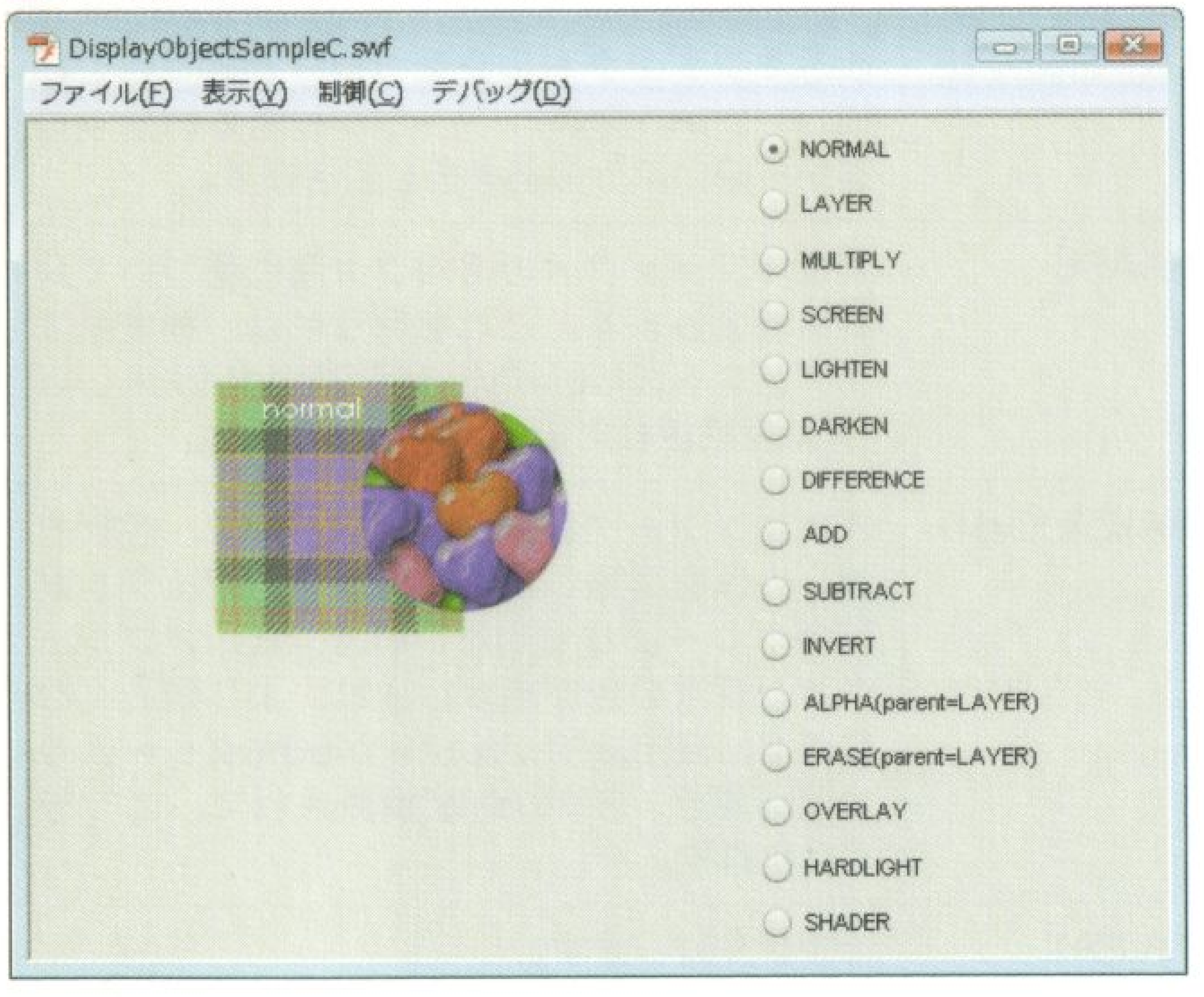
目的	定数名	ブレンドの内容
明るくて刺激的な色の作成	BlendMode.DIFFERENCE	表示オブジェクトの要素カラーと背景色を比較し、2つの要素カラーのうち明るいほうの値から暗いほうの値を差し引きます。
前景のアルファを使用した背景の一部を切り抜く（消す）	BlendMode.ERASE	表示オブジェクトのアルファ値に基づいて背景を消去します。これを行うには、親表示オブジェクトのblendModeがBlendMode.LAYERに設定されている必要があります。
陰影（シャドウ）効果の作成	BlendMode.HARDLIGHT	表示オブジェクトの暗さに基づいて、各ピクセルの色を調整します。表示オブジェクトが50%グレーよりも明るい場合、表示オブジェクトと背景の色が網がけされ、より明るくなります。表示オブジェクトが50%グレーよりも暗い場合、2つの色が乗算されて、より暗くなります。
背景を反転	BlendMode.INVERT	背景を反転します。
特定の表示オブジェクトを事前構成するための一時バッファを強制的に作成	BlendMode.LAYER	表示オブジェクトが処理される前に、一時バッファ内で表示オブジェクトが事前に構成されます。ビットマップキャッシュを使用して表示オブジェクトが事前キャッシュされる際、自動的に実行されます。
重ね合わせ表示	BlendMode.LIGHTEN	表示オブジェクトの要素カラーと背景色のうち明るいほう（値が大きいほう）の色を選択します。
影（シャドウ）と深度効果の作成	BlendMode.MULTIPLY	表示オブジェクトの要素カラーの値と背景色の要素カラーの値を乗算した後、0xFFで割って正規化し、色を暗くします。
陰影（シャドウ）効果の作成	BlendMode.OVERLAY	背景の暗さに基づいて、各ピクセルの色を調整します。
ハイライトとレンズのフレア（表示オブジェクトの黒い領域の削除）	BlendMode.SCREEN	表示オブジェクトの色の補数（逆）と背景色の補数を乗算して、ブリーチ効果を得ます。
ピクセルブレンダシェーダを使ってカスタムブレンド効果を作成	BlendMode.SHADER	カスタムシェーダルーチンを使用して、カラーを調整します。使用するシェーダは、blendShaderプロパティに割り当てられたShaderインスタンスとして指定されます。
2つのイメージ間での色を濃くするディゾルブ効果をアニメーションする	BlendMode.SUBTRACT	下限を0として、表示オブジェクトの要素カラーの値をその背景色の値から減算します。

2
3

PART 1 2 3 4 5 6 7 8 9

SAMPLE

■ NORMAL(デフォルト)



■ HARDLIGHT設定時



SAMPLE CODE

ブレンドモードの変更と確認

DisplayObjectSampleC.fla

```
import flash.display.BlendMode;
import fl.controls.RadioButton;
import fl.controls.RadioButtonGroup;

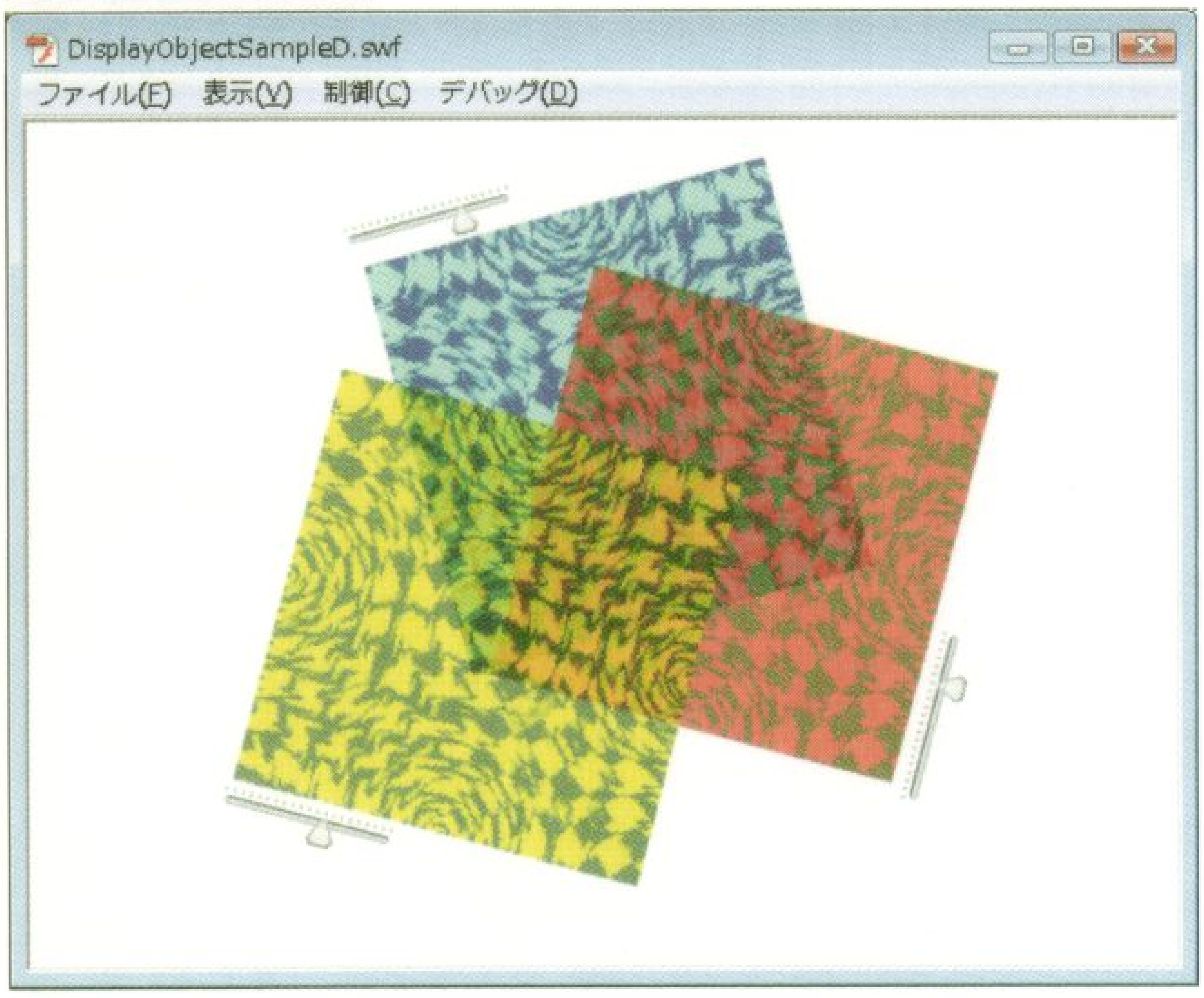
//ブレンドモードを選択するラジオボタンの設定
```

```
var rg:RadioButtonGroup = new RadioButtonGroup( "blendMode" );
rg.addEventListener( Event.CHANGE, setCurrentBlendMode );
for ( var i:int = 0 ; i < this.numChildren ; i++ ) {
  if ( this.getChildAt( i ) is RadioButton ) {
     ( this.getChildAt( i ) as RadioButton ).group = rg;
  }
}                                                              ―①
var currentBlendMode:String = BlendMode.NORMAL;
mode_txt.text = BlendMode.NORMAL;

//円形の表示オブジェクトにブレンドモードを設定する関数
function setCurrentBlendMode(event:Event):void {
  currentBlendMode = BlendMode[ (( event.target as RadioButton
  Group ).selection).value ];
  switch ( currentBlendMode ) {
     //ALPHAとERASEの場合、親がLAYERであることが必要
     case BlendMode.ALPHA:
     case BlendMode.ERASE:
        this.blendMode = BlendMode.LAYER;
        break;                                                 ―②
     default:
        this.blendMode = BlendMode.NORMAL;
        break;
  }
  front_mc.blendMode = currentBlendMode;
  mode_txt.text = currentBlendMode;―――――――――――――③
}
```

①はブレンドモードの選択を行うラジオボタンの初期設定です。②で前面の丸いイメージに選択されたブレンドモードを設定します。ALPHAとERASEの場合、必要により、親のブレンドモードをLAYERにしています。③の1行が表示オブジェクトにブレンドモードを設定する記述です。この行を実行すると画面上の表示オブジェクトのブレンドが行われます。

SAMPLE



SAMPLE CODE

アルファ透明度の変更

DisplayObjectSampleD.fla

```
//イベントリスナーの登録
r_slider.addEventListener(Event.CHANGE, setAlpha );
g_slider.addEventListener(Event.CHANGE, setAlpha );
b_slider.addEventListener(Event.CHANGE, setAlpha );
//アルファ値を変更する関数
function setAlpha(event:Event):void {
  switch ( event.target.name ) {
    case "r_slider":                                   ─┐
      image_r.alpha = r_slider.value;                   │
      break;                                            │
    case "g_slider":                                    │
      image_g.alpha = g_slider.value;                   ├─ ①
      break;                                            │
    case "b_slider":                                    │
      image_b.alpha = b_slider.value;                   │
      break;                                           ─┘
  }
}
```

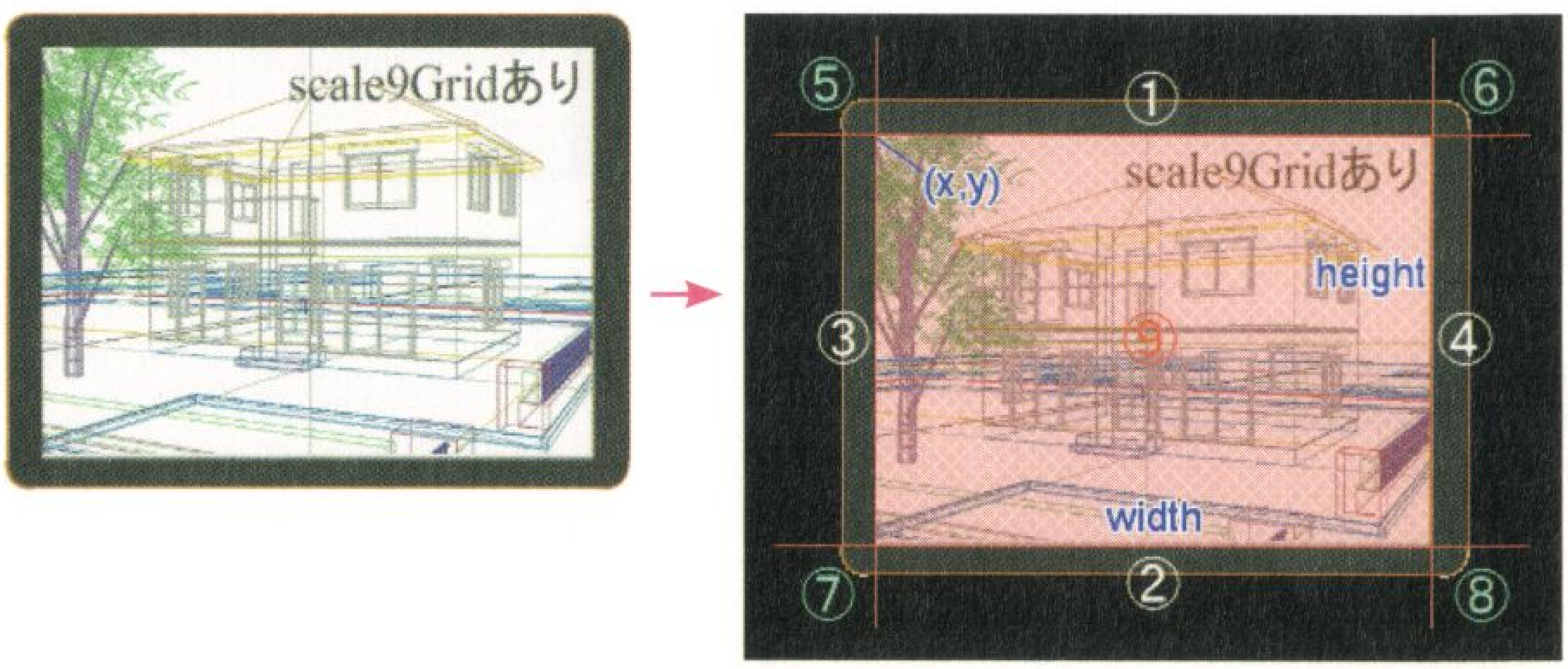
重なり合う3つのイメージのアルファを変更するサンプルです。①で、変更されたスライダーを識別して、表示オブジェクトのアルファ値を変更しています。

7-2-4 表示オブジェクトの9スライス

四角い形状の表示オブジェクトを拡大/縮小するときに4つの角（エッジ）部分は拡大、縮小せずにサイズを維持しておきたいときには、9スライスを設定します。9スライスとは、四角形を上下左右と4つの角、および中央の9部分に分割してそれぞれの拡大、縮小率や方向を調整することで、自然に見えるサイズの変更を実現するものです。例えば角が丸い図形や、額縁のように拡大/縮小したい部分と外枠の部分が分かれているイメージなどを、違和感なくサイズ変更することができます。

DisplayObjectクラスの9スライスに関するプロパティ詳細

<table>
<tr><th>プロパティ</th><th>プロパティの詳細</th></tr>
<tr><td>scale9Grid
:Rectangle</td><td>下図のように9つの部分からなるグリッドを設定して各部分の拡大/縮小の有無と方向を決めます。9つの部分は中心部分⑨を定義するRectangleオブジェクトにより、決定します。Rectangleのx、yは対象の表示オブジェクトのローカル座標、width、heightで幅と高さを表します。



各部分について、次のような規則で拡大/縮小を行います。

<table>
<tr><th>部分</th><th>拡大と縮小</th></tr>
<tr><td>①②</td><td>水平方向にのみ拡大、縮小</td></tr>
<tr><td>②④</td><td>垂直方向にのみ拡大、縮小</td></tr>
<tr><td>⑤⑥⑦⑧</td><td>拡大、縮小なし</td></tr>
<tr><td>⑨</td><td>水平、垂直方向に拡大、縮小</td></tr>
</table>
</td></tr>
</table>

SAMPLE



SAMPLE CODE

表示オブジェクトの9スライスを設定する

DisplayObjectSampleE.fla

```
//scale9Grid用のRectangle
var rect:Rectangle = new Rectangle( 15.5, 14.4, 199.1, 144 ); ——①
//scale9Gridプロパティに作成したRectangleオブジェクトを設定
image2_mc.scale9Grid = rect; ——②
//拡大縮小のためのスライダー設定
scale_slider.addEventListener( Event.CHANGE, setScale );
function setScale(event:Event):void {
  image_mc.scaleX = event.target.value;
  image_mc.scaleY = event.target.value;
  image2_mc.scaleX = event.target.value;
  image2_mc.scaleY = event.target.value;
}
```

①でRecangleオブジェクトを作成し、②で表示オブジェクトに設定しています。上記の実行結果の写真は左がオリジナルサイズ、右が2倍に拡大したものです。scale9Gridを設定した場合、メイン画像の周りの「ふち」の部分が元と同じ幅を維持することがわかります。

7-3 インタラクティブオブジェクト（InteractiveObject）

7-3-1 インタラクティブオブジェクトのイベントとプロパティを使う

表示オブジェクトに対するマウス、キーボードやフォーカスに関する操作を処理するには、イベントの監視が基本となります。イベントを監視するための仕組みはDisplayObjectクラスを基本クラスとするInteractiveObjectクラスで提供されます。

InteractiveObjectクラスの主なプロパティ概要

目的	プロパティ名	プロパティの概要
ダブルクリックイベントを受け取る/受け取らないを指定	doubleClickEnabled:Boolean	オブジェクトがダブルクリックイベントを受け取るかを決定します。
マウスイベントを受け取る/受け取らないかを指定	mouseEnabled:Boolean	マウスイベントを受け取るかを決定。
フォーカス矩形を表示する/しないを指定	focusRect:Object	フォーカスを得たときに矩形を表示するかを決定します。
タブ順序に含める/含めないを指定	tabEnabled:Boolean	タブ移動順に含まれるかを決定します。
タブ移動順を指定	tabIndex:int	タブ移動順に含まれる場合、その順番を決定します。

InteractiveObjectクラスの主なイベント概要

タイミング	イベント名	イベントの概要
マウスのボタンを押して離したとき	click mouseDown mouseUp	表示オブジェクトの上でマウスボタンをクリックするとmouseDown、続いてボタンを離すとmouseUp、その後clickが発生します。
マウスのボタンをダブルクリックしたとき	doubleClick	マウスボタンを2回続けてクリックするとこのイベントが送出されます。
マウスを移動させたとき	mouseMove	ユーザがマウスカーソルをInteractiveObject上で動かしている間、mouseMoveイベントが連続して発生します。
マウスカーソルがオブジェクト内に入ったとき/出たとき	mouseOver rollOver mouseOut rollOut	マウスカーソルがInteractiveObjectの外部から領域内に入ると、rollOver、mouseOverイベントが送出されます。マウスカーソルがInteractiveObjectから外に移ると、mouseOut、rollOutイベントが送出されます。
マウスホイールを回転させたとき	mouseWheel	InteractiveObject上でマウスホイールを回転させたとき、このイベントが送出されます。
マウスによりオブジェクトがフォーカスを取得/喪失したとき	mouseFocusChange	mouseFocusChangeイベントは現在フォーカスを持っているオブジェクトがマウス操作によってフォーカスを変更される直前に送出されます。

7-3-2 マウスイベントを処理する

PART
1
2
3
4
5
6
7
8
9

表示オブジェクトに対するマウス操作をActionScriptで操作するには、InteractiveObjectクラスで提供されるイベント処理を利用します。インタラクティブな操作を行うDisplayObjectContainer、SimpleButton、TextField、Loader、Sprite、Stage、MovieClipなどのクラスはすべてInteractiveObjectを基本として作成されているため、マウスに関するイベントの処理方法は共通になります。オブジェクトがマウスイベントを受け取る/受け取らないのコントロールは、InteractiveObject.mouseEnabled、doubleClickEnabledプロパティで制御することができます。マウスに関するイベント名はMouseEventクラスに定数が用意されています。また、マウスイベントが発生するとやはりMouseEventクラスのオブジェクトが生成され、情報が格納されます。

InteractiveObjectクラスのマウス処理に関するプロパティ詳細

プロパティ	プロパティの詳細
mouseEnabled :Boolean	オブジェクトがマウスに関するイベントメッセージを受けるかを決定します。ただし、このオブジェクトに子がある場合、その子がマウスイベントを受け取るかまでは、親のプロパティではコントロールできません。子オブジェクトのマウス処理を制限するには、DisplayObjectContainerクラスにあるmouseChildrenプロパティをfalseにします。 **true**：オブジェクトはマウスイベントを受け取ります(デフォルト)。 **false**：オブジェクトはマウスイベントを受け取りません。
doubleClick Enabled:Boolean	デフォルトではInteractiveObjectはdoubleClickイベントの受け取りをしないため、ダブルクリック処理をするには、このプロパティをtrueにしておく必要があります。また、mouseEnabledプロパティがfalseになっているとdoubleClickイベントも通知されないので、こちらもtrueにしておく必要があります。 **true**：オブジェクトはダブルクリックイベントを受け取ります。 **false**：オブジェクトはダブルクリックイベントを受け取りません(デフォルト)。

InteractiveObjectクラスのマウスに関するイベント詳細

イベント(下段はMouseEventクラス定数)	イベントの詳細
click mouseDown mouseUp CLICK MOUSE_DOWN MOUSE_UP	ユーザがオブジェクトの上でマウスボタンをクリックするとmouseDownイベント、続いてボタンを離すとmouseUpイベント、その後clickイベントが発生します。オブジェクト内でmouseDownを発生させてボタンを押したままオブジェクト領域を外れてmouseUpした場合、clickイベントは発生しません。同じように、オブジェクト外でmouseDown、その後ボタンを押したままオブジェクト領域でmouseUpの場合もclickは発生しません。 つまり、オブジェクト内で確実にマウスボタンを押して離した場合のみ、clickが発生します。

InteractiveObjectクラスのマウスに関するイベント詳細（続き）

イベント（下段はMouseEventクラス定数）	イベントの詳細
doubleClick	オブジェクト上のダブルクリックを知らせるイベントです。doubleClickが発生するためには、オブジェクト領域内でclickが2回続けて発生する必要があります。 このイベントを処理するには、doubleClickEnabledプロパティをtrueにしておく必要があります。
DOUBLE_CLICK	
mouseMove	オブジェクト上でユーザがマウスポインタを移動したときに継続的に発生します。
mouseOver rollOver mouseOut rollOut	マウスカーソルがInteractiveObjectの外部から領域内に入ると、rollOverイベントに続いてmouseOverイベントが送出されます。そのオブジェクト内の子InteractiveObjectにカーソルが移動し、再び領域内に入ったとき、mouseOverイベントのみが送出されます。 マウスカーソルがInteractiveObjectから外に移ると、mouseOutイベントに続いてrollOutイベントが送出されます。 mouseOutイベントは子InteractiveObjectにカーソルが移動するときにも発生します。
MOUSE_OVER ROLL_OVER MOUSE_OUT ROLL_OUT	
mouseWheel	オブジェクト上でマウスホイールを回転させたときに発生します。
MOUSE_WHEEL	
mouseFocusChange	オブジェクトが現在フォーカスを持っていて、ユーザのマウス操作によりフォーカスを失おうとするときに発生します。フォーカスを失った後はfocusOutイベントが発生します（FocusEventクラス）。
MOUSE_FOCUS_CHANGE	

MouseEventクラスのマウスイベントに関するプロパティ

目的	プロパティ名	プロパティの概要
Altキーが押されているか	altKey :Boolean	**true** ：Altキーが押されています。 **false**：Altキーは押されていません。
マウスの主ボタンが押されているか	buttonDown :Boolean	**true** ：主ボタンが押されています。 **false**：主ボタンは押されていません。
Ctrlキーが押されているか（Windows/Linux）	ctrlKey :Boolean	**true** ：Ctrlキーが押されています。 **false**：Ctrlキーは押されていません。
マウスホイールの1目盛りでスクロールする行数	delta :int	正のdelta値は上方向、負の値は下方向へのスクロールを表します。ホイールの回転が速くなると、deltaの値は大きくなります。設定値は、デバイスやオペレーティングシステムで異なります。
マウスイベントの発生位置（ローカル座標）	localX localY :Number	表示チェーンの最下位の子でのクリック位置座標を示します。
Shiftキーが押されているか	shiftKey :Boolean	**true** ：Shiftキーが押されています。 **false**：Shiftキーは押されていません。
マウスイベントの発生位置（グローバル座標）	stageX stageY :Number	ステージを基準としたクリック位置座標を示します。

SAMPLE



このサンプルは、イメージ上で発生したマウスイベント名をすべて表示するものです。

SAMPLE CODE

マウスクリックイベントを処理する

IntractiveObjectSampleA.fla

```
import fl.managers.FocusManager;
                                                              ①
var fm:FocusManager = new FocusManager( this );

//イメージに対するマウスイベントハンドラを設定
Coins_mc.addEventListener( MouseEvent.CLICK, showEvent );
Coins_mc.addEventListener( MouseEvent.MOUSE_DOWN, showEvent );
Coins_mc.addEventListener( MouseEvent.DOUBLE_CLICK, showEvent );
Coins_mc.addEventListener( MouseEvent.MOUSE_DOWN, showEvent );
Coins_mc.addEventListener( MouseEvent.MOUSE_MOVE, showEvent );
Coins_mc.addEventListener( MouseEvent.MOUSE_OUT, showEvent );      ②
Coins_mc.addEventListener( MouseEvent.MOUSE_OVER, showEvent );
Coins_mc.addEventListener( MouseEvent.MOUSE_UP, showEvent );
Coins_mc.addEventListener( MouseEvent.MOUSE_WHEEL, showEvent );
Coins_mc.addEventListener( MouseEvent.ROLL_OUT, showEvent );
Coins_mc.addEventListener( MouseEvent.ROLL_OVER, showEvent );

//doubleClickEnabledチェックボックスのイベントハンドラ
dblClickEnabled_chk.addEventListener( MouseEvent.CLICK,
```

```
setDblClickEnabled );
mouseEnabled_chk.addEventListener( MouseEvent.CLICK, setMouseEnabled
);

//テキストの選択範囲を示すための変数
var sidx:int, lidx:int;
function showEvent( event:Event ):void {
  sidx = message_txt.length;
  //event.type でイベント名を取り出して出力する
  message_txt.appendText( event.type ); ――③
  lidx = message_txt.length;
  message_txt.appendText( "¥n" );
  fm.setFocus( message_txt );
  message_txt.setSelection( sidx, lidx );
}

function setDblClickEnabled(event:Event):void {
  //イメージがダブルクリックイベントを受け取る/受け取らないように設定
  Coins_mc.doubleClickEnabled = dblClickEnabled_chk.selected; ――④
}

function setMouseEnabled(event:Event):void  {
  //マウスイベントを処理する/処理しないを設定
  Coins_mc.mouseEnabled = mouseEnabled_chk.selected; ――⑤
}
```


①はTextAreaにフォーカスを与える処理で、マウスイベント処理とは関係ありません。②でInteractiveObject（ここではイメージをシンボル化したMovieClip）で処理できるすべてのマウスイベントをshowEvent()関数でハンドルするようにしています。この関数では③で使用している、イベントの名前を表すEventオブジェクトのtypeプロパティだけが重要です。この場所以外の処理はTextAreaに追加したテキストを選択状態にすることを行っているだけです。

動かすとわかりますが、このままではダブルクリックイベントは処理されません。チェックボックスのオン/オフでダブルクリックイベントを受け取るようにdoubleClickEnabledプロパティを設定しているのが④です。⑤で行っているように、mouseEnabledプロパティをtrue(デフォルト）/falseにすることで、マウスイベントを受け取る/受け取らないを変更できます。

7-3-3 キーボードイベントを処理する

PART
1
2
3
4
5
6
7
8
9

表示オブジェクトに対するキーボード操作を処理するための仕組みは、InteractiveObjectクラスで提供されます。インタラクティブな操作を行うDisplayObjectContainer、SimpleButton、TextField、Loader、Sprite、Stage、MovieClipなどのクラスはすべてこのInteractiveObjectを基本として作成されているため、マウス処理と同じようにキーボードに関するイベント処理は共通になります。キーボードイベントが発生すると、KeyboardEventクラスのオブジェクトが生成され、情報が格納されます。OSや Web ブラウザは、Flash Playerより先にキーボードイベントを処理するため、例えばWindowsでは、Alt + Tab キーを押したとき、Flash Playerがキーボードイベントを送出する前に、ブラウザから他のアプリケーションにウィンドウを移す操作になります。

InteractiveObjectクラスのキーボードに関するイベント詳細

イベント（下段は定数）	イベントの詳細
keyFocusChange KEY_FOCUS_CHANGE	ユーザのキーボード操作により現在のオブジェクトがフォーカスを失おうとするときに発生します。フォーカスを失った後はfocusOutイベントが発生します。フォーカスが移動した先のオブジェクトではfocusInイベントが発生します。この定数はFocusEventクラスで定義されています。
keyDown keyUp KEY_DOWN KEY_UP	オブジェクトがフォーカスを得た状態でキーボードが押されるとkeyDownイベントが発生します。押し続けている場合、連続的にイベントが発生します。キーを離すとkeyUpイベントが発生します。StageにkeyDown/keyUpイベントリスナーを設定すると、アプリケーション稼働中に発生したすべてのキーボードイベントを処理できます。 イベントオブジェクトとしてKeyboardEventオブジェクトが作成されますが、そのプロパティでcommand/ctrl/alt/shiftキーなどの情報を調べることができます。 キーイベントを表す定数はKeyboardEventクラスで定義されています。

KeyboardEventクラスのマウスイベントに関するプロパティ

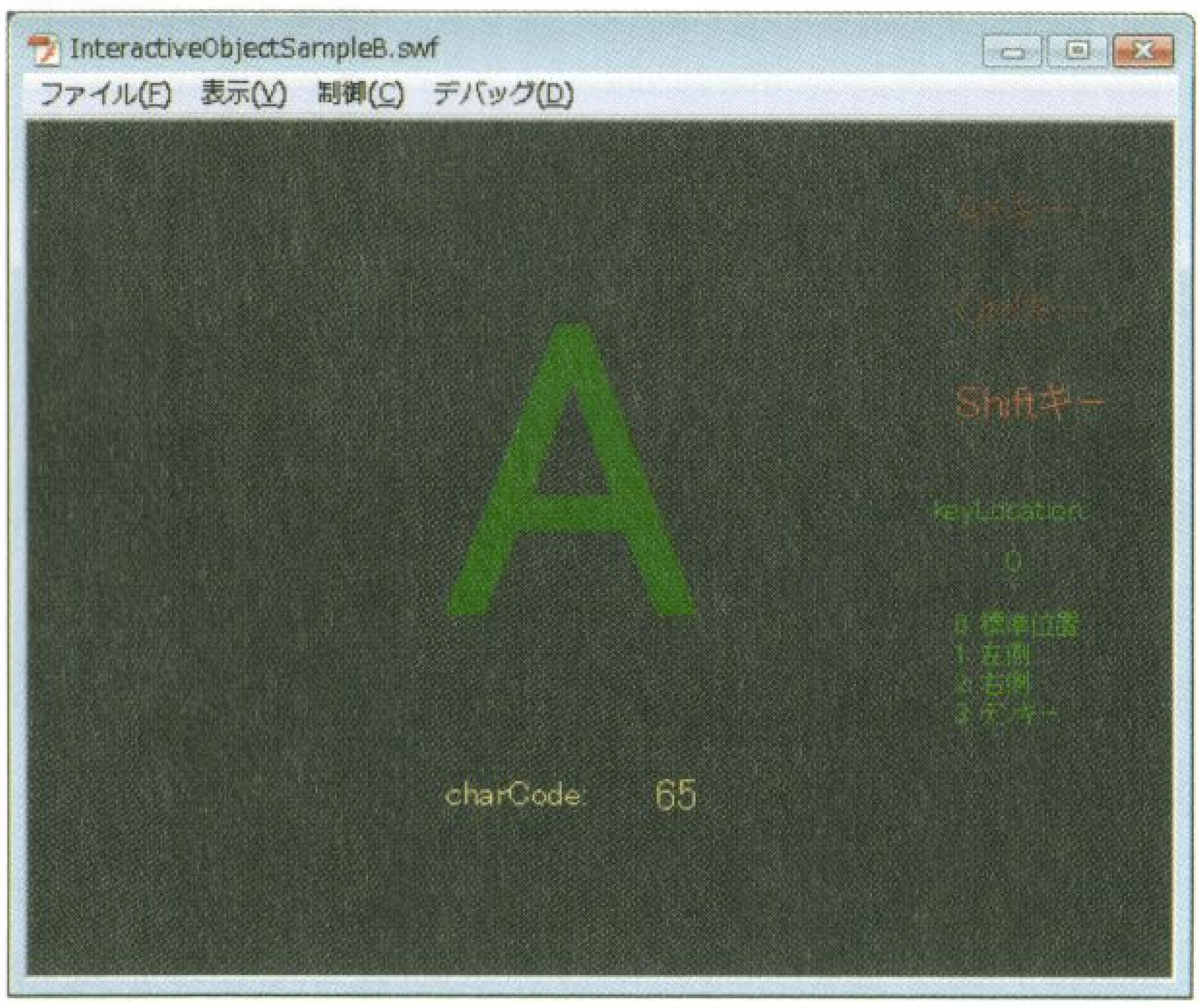
目的	プロパティ名	プロパティの概要
Altキーが押されているか	altKey :Boolean	**true** ：Altキーが押されています。 **false**：Altキーは押されていません。
どのキーが押されたか	charCode :uint	押されたキー（離されたキー）の文字コード値を取得できます。
Ctrl（Windows/Linux）/ Command（Mac OS）キーが押されているか	ctrlKey :Boolean	**true** ：Ctrl/Commandキーが押されています。 **false**：Ctrl/Commandキーは押されていません。
キーコード値	keyCode :uint	押されたキー（離されたキー）のキーコード値を取得できます。キーコードはキーボード上のキーを識別するための数値コードです。

KeyboardEventクラスのマウスイベントに関するプロパティ（続き）

<table>
<tr><th>目的</th><th>プロパティ名</th><th>プロパティの概要</th></tr>
<tr><td>キーボードのキーの場所</td><td>key
Location
:uint</td><td>「2つあるShiftキーのうち左右どちらが押されたか」、「数値がテンキーで入力されたか」など、同一文字で2種類のキーがある場合の識別情報を提供します。KeyLocationクラスに次の定数が用意されています。
<table>
<tr><th>定数名（uint値）</th><th>意味</th></tr>
<tr><td>STANDATD(0)</td><td>キーの区別がないか、テンキーでないキーが押されました。</td></tr>
<tr><td>LEFT(1)</td><td>左側にあるキーが押されました。</td></tr>
<tr><td>RIGHT(2)</td><td>右側にあるキーが押されました。</td></tr>
<tr><td>NUM_PAD(3)</td><td>テンキーが押されました。</td></tr>
</table></td></tr>
<tr><td>Shiftキーが押されているか</td><td>shiftKey
:Boolean</td><td>true ：Shiftキーが押されています
false：Shiftキーは押されていません</td></tr>
</table>

3
3

SAMPLE



SAMPLE CODE

キーボードが押された/離されたイベントと押されたキーを取得する

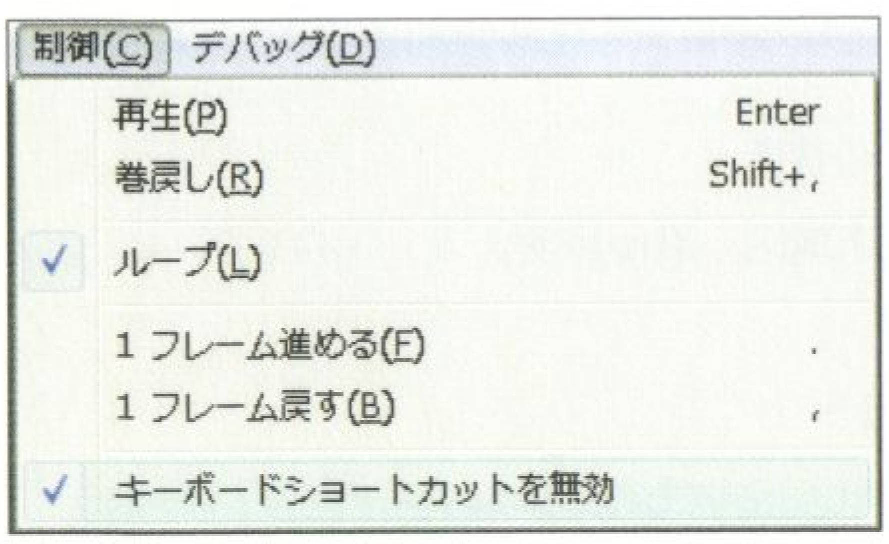
IntractiveObjectSampleB.fla

```
//ステージでキーボードイベントを取得するためのイベントハンドラの登録
stage.addEventListener( KeyboardEvent.KEY_DOWN, showKey );  ①
//KeybordEventを処理するハンドラ
function showKey( event:KeyboardEvent ):void {
  char_txt.text = String.fromCharCode( event.keyCode ) ;  ②
  //keyCodeの表示
```

```
  keycode_txt.text =  event.keyCode.toString() ;
  //文字のアルファ値を元に戻す
  alt_txt.alpha = 0.3; shift_txt.alpha = 0.3; ctrl_txt.alpha = 0.3;
  //Altキーの判定
  if ( event.altKey ) {
    alt_txt.alpha = 100;
  }
  //Shiftキーの判定
  if ( event.shiftKey ) {
    shift_txt.alpha = 100;
  }
  //Ctrlキーの判定
  if ( event.ctrlKey ) {
    ctrl_txt.alpha = 100;
  }
  //keyLocationの判定
  keylocation_txt.text = event.keyLocation.toString();――――③
}
```

①のコードにより、ステージ上のキー入力をshowKey()関数で処理できます。②でKeyboardEventオブジェクトのkeyCodeプロパティで取得したキーコードを文字に変更しています。keyCodeプロパティはIMEがオンのときは正常なコードを返しません。つまり、Ascii文字のみをサポートします。③で行っているkeyLocationプロパティにより、キーボードの左側/右側どちらのCtrlキーが押されたか、テンキーが押されたかなどが判定できます。

ムービープレビューのときにCtrl + Sなどのキー入力はショートカットとして登録されており、アプリケーションより先に処理されてしまうため、キャプチャーされません。このようなキー操作を判定するには、プロジェクタの[制御]メニューの[キーボードショートカットを無効]にチェックをつけます。ただし、Altキーはプロジェクタ自体のメニューを選択してしまうため、アプリケーションではキャプチャーできません。



7-3-4 特殊キーの入力を処理する

flash.ui.Keyboardクラスのプロパティにより、capsLock、numLockキーが現在アクティブになっているかを識別できます。また、同クラスにはスペースキーやEnterキーなど、文字コードではなくキーコードで識別する際に使用できる定数が用意されています。capsLock、numLockキーをActionScriptでアクティブ/非アクティブに変更することはできません。Keyboardクラスのプロパティと定数はすべて静的で、インスタンスを生成する必要はありません。

Keyboardクラスの特殊キーマウスイベントに関するプロパティ

目的	プロパティ名	プロパティの概要
capsLockキーがアクティブか	capsLock :Boolean	**true**：現在CapsLockキーがアクティブです。 **false**：現在CapsLockキーが非アクティブです。 読み取り専用で、値を変更することはできません。
numLockキーがアクティブか	numLock :Boolean	**true**：現在NumLockキーがアクティブです。 **false**：現在NumLockキーが非アクティブです。 読み取り専用で、値を変更することはできません。

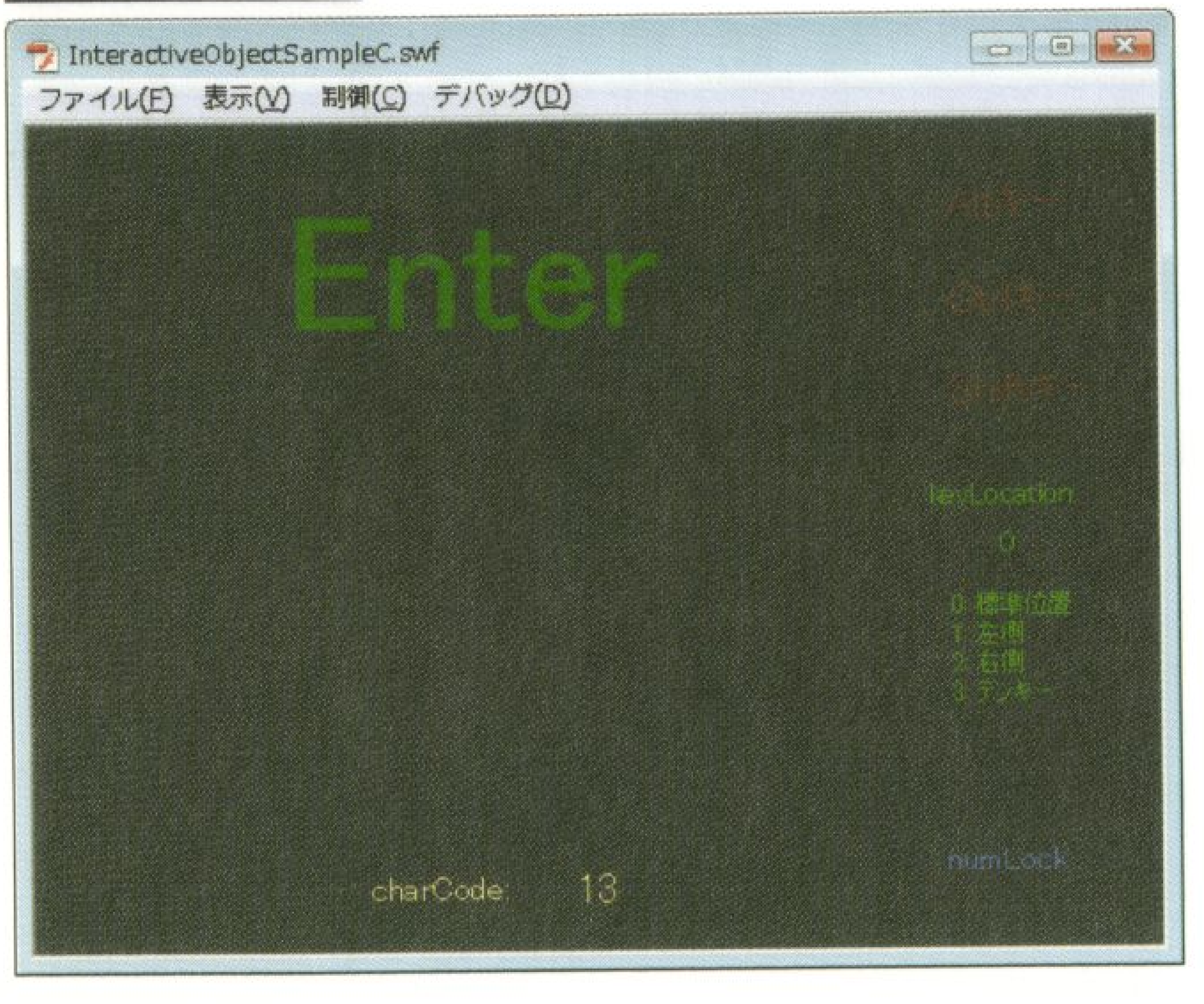
Keyboardクラスの特殊キーのキーコードを表す定数

目的のキー	定数	キーコード値
スペースキー	SPACE	32
Enterキー	ENTER	13
バックスペースキー	BACKSPACE	8
Ctrlキー	CONTROL	17
Deleteキー	DELETE	46
Endキー	END	35
Escキー	ESCAPE	27
F1～F15	F1～F15	112～126
HOMEキー	HOME	36
Insキー	INSERT	45
テンキーの0～9	NUMPAD_0～NUMPAD_9	96～105
テンキーの次のキー + . / Enter * -	 NUMPAD_ADD NUMPAD_DECIMAL NUMPAD_DIVIDE NUMPAD_ENTER NUMPAD_MULTIPLY NUMPAD_SUBTRACT	 107 110 111 108 106 109

Keyboardクラスの特殊キーのキーコードを表す定数(続き)

目的のキー	定数	キーコード値
PageDownキー	PAGE_DOWN	34
PageUpキー	PAGE_UP	33
Shiftキー	SHIFT	16
Tabキー	TAB	9
矢印キーの上(↑)	UP	38
矢印キーの下(↓)	DOWN	40
矢印キーの左(←)	LEFT	37
矢印キーの右(→)	RIGHT	39

SAMPLE



SAMPLE CODE

特殊キーを識別する

IntractiveObjectSampleC.fla

```
import flash.ui.Keyboard;
//ステージでキーボードイベントを取得するためのイベントハンドラの登録
stage.addEventListener( KeyboardEvent.KEY_DOWN, showKey );
//KeyboardEventを処理するハンドラ
function showKey( event:KeyboardEvent ):void {
  char_txt.text = String.fromCharCode( event.keyCode ) ;
  //keyCodeの表示
  keycode_txt.text =  event.keyCode.toString() ;
```

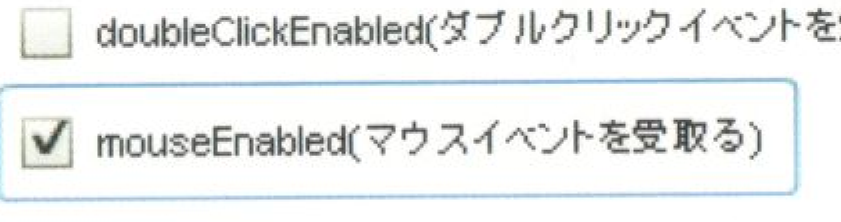
```
    //文字のアルファ値を元に戻す
    alt_txt.alpha = 0.3; shift_txt.alpha = 0.3; ctrl_txt.alpha = 0.3;
    //Altキーの判定
    if ( event.altKey ) {
      alt_txt.alpha = 100;
    }
    //Shiftキーの判定
    if ( event.shiftKey ) {
      shift_txt.alpha = 100;
    }
    //Ctrlキーの判定
    if ( event.ctrlKey ) {
      ctrl_txt.alpha = 100;
    }
    //keyLocationの判定
    keylocation_txt.text = event.keyLocation.toString();
    //特殊キー (スペース、Enter) の判定
    special_txt.text = "";
    switch( event.keyCode ) {
      case Keyboard.SPACE:
        special_txt.text = "Space";break;
      case Keyboard.ENTER:                                  ①
        special_txt.text = "Enter";break;
    }
    checkSpecialKeys();
}
//capsLock, numLockキーの判定
function checkSpecialKeys():void {                          ②
    capslock_txt.visible = Keyboard.capsLock;
    numlock_txt.visible = Keyboard.numLock;
}
//初期値は非表示にする
capslock_txt.visible = false;
numlock_txt.visible = false;
```

7-3-3 のサンプルではスペースキーやEnterキーなどの特殊キーは表示されません。ここではKeyboardクラスの定数により、charCodeを判定し、スペースキーとEnterキーの場合、文字を表示します。判定は①で行っています。また、capsLockキーとnumLockキーが押されているかを②から始まるcheckSpecialKeys()関数で判定しています。

7-3-5 Tabによるフォーカス移動を制御する

表示オブジェクトに対するフォーカスの変化に関する処理を行うための仕組みも、DisplayObjectクラスを基本クラスとするInteractiveObjectクラスで提供されます。インタラクティブな操作を行うDisplayObjectContainer、SimpleButton、TextField、Loader、Sprite、Stage、MovieClipなどのクラスはすべてInteractiveObjectを基本として作成されているため、フォーカスに関するイベント処理は共通になります。また、DisplayObjectContainerクラスには子オブジェクトがTabによるフォーカスを受ける/受けないを決定するtabChildrenプロパティがあります。

InteractiveObjectクラスのフォーカスに関するプロパティ詳細

プロパティ	プロパティの詳細
focusRect :Boolean	オブジェクトがフォーカスを受けたときにそのことを示す枠を表示するかを決定します。次の2つのチェックボックスでは、下のオブジェクトがフォーカス枠を表示しています。  **true** ：フォーカス枠を表示します（デフォルト） **false**：フォーカス枠を表示しません。
tabEnabled :Boolean	オブジェクトがTabキーによるフォーカス移動順序に含まれるかを決定します。 **true** ：オブジェクトはタブ移動順に含まれます(次のケースでデフォルト値=trueです)。 ・SimpleButtonオブジェクト ・TextFieldでtype = "input"の場合 ・Sprite/MovieClipオブジェクトでbuttonMode=trueの場合 **false**：オブジェクトはタブ移動順に含まれません。（上記のケース以外ではデフォルト値= falseです）
tabIndex :int	Tabキーによるフォーカス移動順序を決定する整数値です。値が小さいほどフォーカスを受ける順序が早くなります。デフォルト値は-1で、Tabインデックスが設定されていないことを表します。アプリケーション内のすべての表示オブジェクトのtabIndexがデフォルト値の場合、自動的にタブ移動順が割り当てられますが、1つでもtabIndex値が設定されたオブジェクトがあると、自動タブ移動順は無効になり、tabIndexに値が設定されたオブジェクトだけがTabキーによるフォーカス移動の対象となります。

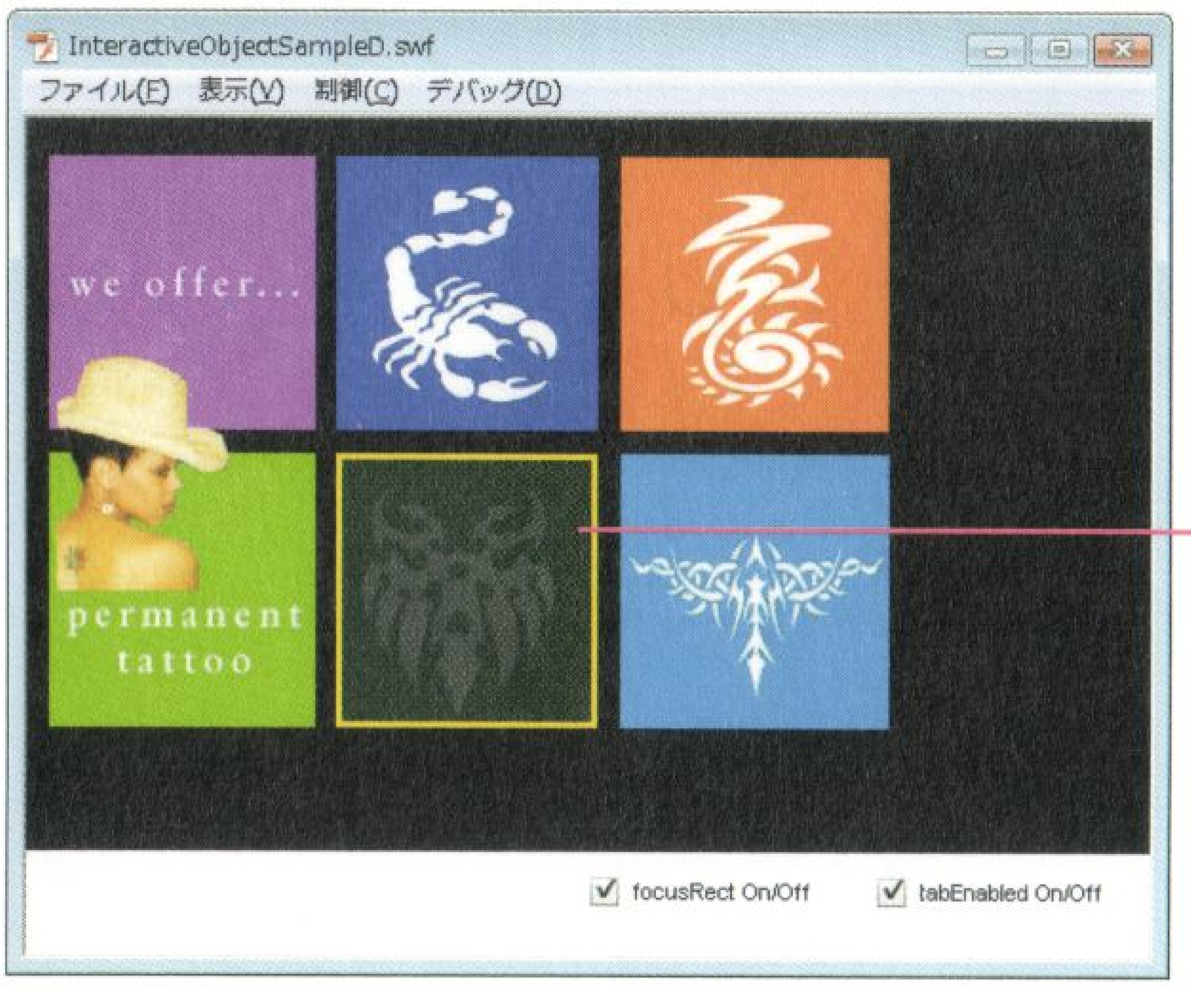
InteractiveObjectクラスの主なイベント詳細

イベント名 (下段：FocusEventクラス定数)	イベントの詳細
focusIn focusOut keyFocusChange mouseFocusChange	オブジェクトがフォーカスを取得してハイライトされるときにfocusInイベントが発生し、フォーカスを喪失した後、focusOutイベントが発生します。 オブジェクトが現在フォーカスを持っていて、ユーザのキーボード操作によりフォーカスを失おうとするときはkeyFocusChangeイベントが発生します。その後フォーカスを失った後はfocusOutイベントが発生し、フォーカスを得たオブジェクトでfocusInが発生します。
FOCUS_IN FOCUS_OUT KEY_FOCUS_CHANGE MOUSE_FOCUS_CHANGE	

DisplayObjectContainerクラスのフォーカスに関するプロパティ詳細

プロパティ	プロパティの詳細
tabChildren :Boolean	DisplayObjectContainerオブジェクトが持つ子オブジェクトがTabによるフォーカスを受けるか、受けないかを決定します。 **true** ：フォーカスを受けます(デフォルト)。 **false** ：フォーカスを受けません。

SAMPLE



focusRect

SAMPLE CODE

DisplayObjectのフォーカスプロパティとイベント　InteractiveObjectSampleD.fla

```
var i:int;

//tabIndexを設定
function setTabIndex():void {
```

```
  g1.tabIndex=1;
  g2.tabIndex=2;
  g3.tabIndex=3;
  g4.tabIndex=4;
}

//MovieClipにtabEnabled, focusRect, イベントハンドラを設定
function setEventHandler():void {
  for ( i = 1 ; i <= 4 ; i++ ) {
    this[ "g" + i ].addEventListener( FocusEvent.FOCUS_IN, setFcs );
    this[ "g" + i ].addEventListener( FocusEvent.FOCUS_OUT, setFcs );
  }
  setTabEnabled_chk.addEventListener( Event.CHANGE, setTabEnabled );
  setFocusRect_chk.addEventListener( Event.CHANGE, setFocusRect );
  setTabEnabled_chk.tabEnabled = false;
  setFocusRect_chk.tabEnabled = false;
}

//tabEnabedプロパティの設定
function setTabEnabledSettings( enableTab:Boolean ):void {
  for ( i = 1 ; i <= 4 ; i++ ) {
    this[ "g" + i ].tabEnabled=enableTab;        ①
  }
}

//focusRectプロパティの設定
function setFocusRectSettings( enableRect:Boolean ):void {
  for ( i = 1 ; i <= 4 ; i++ ) {
    this[ "g" + i ].focusRect=enableRect;        ②
  }
}

//tabEnabed On/Offチェックボックスのイベントハンドラ
function setTabEnabled(event:Event):void {
  setTabEnabledSettings( setTabEnabled_chk.selected );
  //tabEnabled = false; により、focusRectもfalseになるため、trueに戻します
  if ( setFocusRect_chk.selected ) setFocusRectSettings(
  setFocusRect_chk.selected );
}
```

```
//focusRect On/Offのイベントハンドラ
function setFocusRect(event:Event):void {
  setFocusRectSettings( setFocusRect_chk.selected );
}

//focusIn / focusOutのイベントハンドラ
function setFcs( event:FocusEvent ):void {
  switch ( event.type ) {
    case FocusEvent.FOCUS_IN :
      event.target.alpha=0.2;
      break;                                    ③
    case FocusEvent.FOCUS_OUT :
      event.target.alpha=1.0;
      break;
  }
}

//初期呼び出し
setTabIndex();
setEventHandler();
setTabEnabledSettings( true );
setFocusRectSettings( true );
```

6つのムービークリップのうち、右側の4つに対して、タブでフォーカス移動するサンプルです。画面下部のfocusRect On/OffチェックボックスでfocusRect、tabEnabled On/Offチェックボックスでt abEnabledプロパティの変更を行います。それらは①で4つのムービークリップのtabEnabledプロパティを、②でfocusRectプロパティを設定しています。③でfocusIn, focusOutイベントの処理を行い、フォーカスを得たムービークリップのアルファ値を下げています。ここでは、event.typeでイベントの種別を判定しています。このサンプルをムービープレビューするときは、[制御]-[キーボードショートカットを無効]メニューにチェックをつけます。

7 - 4 FocusManager

7 - 4 - 1 FocusManagerを使う

Tab移動順によるフォーカス移動ではなく、ActionScriptによるフォーカスの制御を行う場合はFocusManagerクラスを使用します。FocusManagerはDisplayObjectContainer単位で、含まれる子オブジェクトをグループ化（タブグループ）してフォーカス管理することができます。例えばDisplayObjectContainerコンテナが2つある場合、2つのFocusManagerオブジェクトを作成し、それぞれでコンテナに含まれる子オブジェクトグループのフォーカス制御を行うことができます。getFocus()メソッドで現在フォーカスを持っているオブジェクトを取得できます。またプログラムでフォーカスを移動するには、setFocus()メソッドを使用します。

FocusManagerクラスの主なプロパティ

目的	プロパティ名	プロパティの概要
デフォルトのボタンを設定/取得する	defaultButton :Button	現在設定されているデフォルトボタンを表します。
デフォルトのボタンが有効か	defaultButtonEnabled :Boolean	デフォルトボタンの有効/無効を決定します。

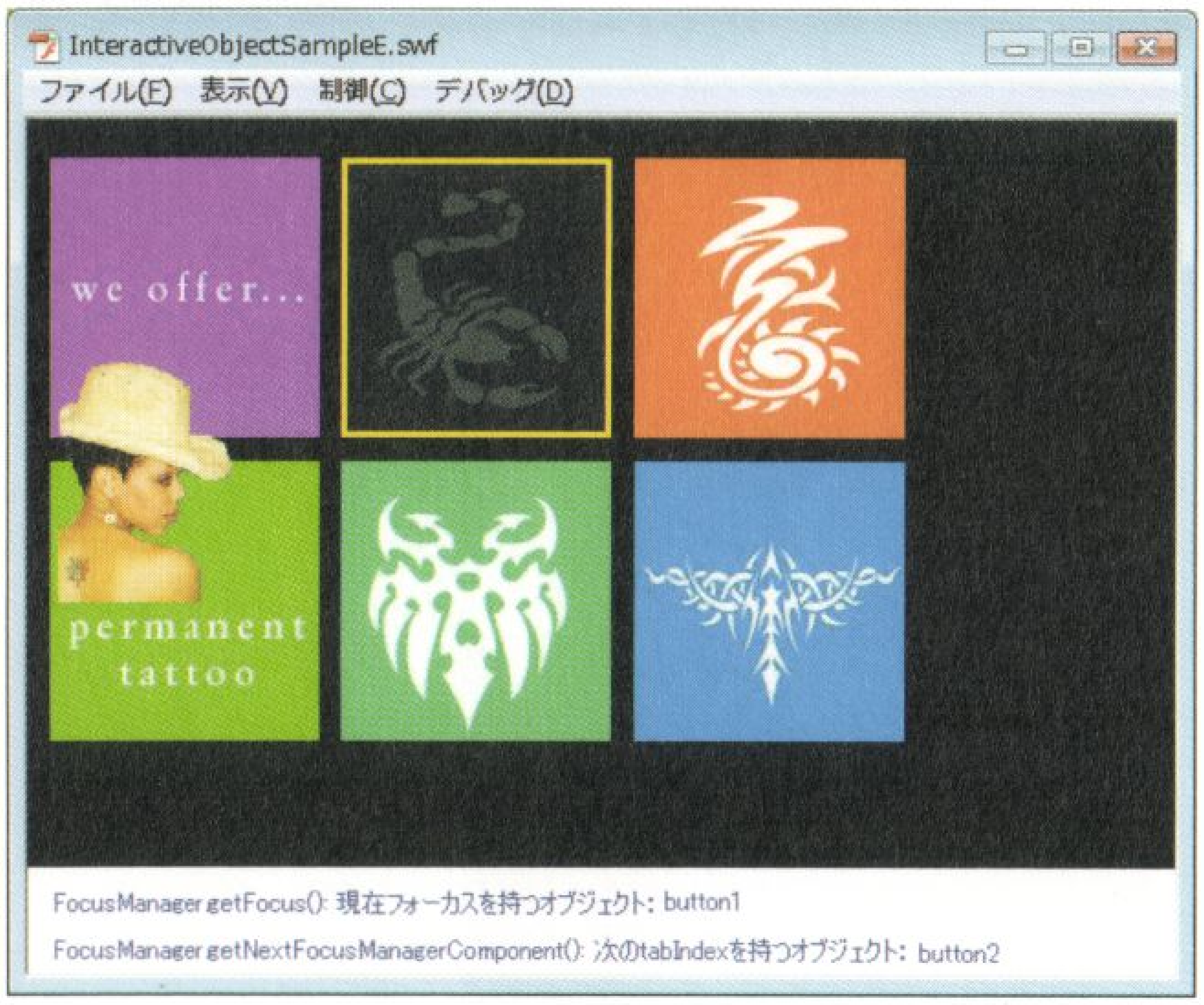
FocusManagerクラスの主なメソッド

目的	メソッド名	メソッドの概要
管理する対象のDisplayObjectContainerオブジェクトを指定する	FocusManager(container: DisplayObject Container)	コンストラクタです。引数に対象とするDisplayObjectContainerオブジェクトを指定します。
FocusManagerをアクティブ/非アクティブにする	activate():void deactivate():void	FocusManagerインスタンスをアクティブ/非アクティブにしてフォーカス制御を行えるようにします。
フォーカスを持っているオブジェクトを調べる	getFocus(): InteractiveObject	現在フォーカスを持っているオブジェクトを取得します。
Tabキーで次にフォーカスを取得するオブジェクトを調べる	getNextFocusManager Component(backward:Boolean = false):InteractiveObject	Tabキーにより、次にフォーカスを受けるオブジェクトがある場合、取得できます。
フォーカスインジケータを削除/表示する	hideFocus():void showFocus():void	現在フォーカスを持っているオブジェクトがインジケータを持つ場合、削除/表示します。
指定したオブジェクトにフォーカスを移す	setFocus(component: InteractiveObject):void	指定したオブジェクトにフォーカスを移します。

FocusManagerクラスのフォーカス移動/取得に関するメソッド詳細

メソッド	メソッドの詳細
getFocus ():InteractiveObject	FocusManagerが管理するInteractiveObjectContainerインスタンスの中で、現在フォーカスを持っているオブジェクトを返します。
setFocus (component:InteractiveObject):void	パラメータで指定したオブジェクトにフォーカスを移します。
getNextFocusManagerComponent (backward:Boolean = false): InteractiveObject	Tabキーを押したときに次にフォーカスを取得するオブジェクトがあれば、そのオブジェクトを返します。次の有効なオブジェクトがない場合、現在フォーカスを持ったオブジェクトが返されます。

SAMPLE



SAMPLE CODE

FocusManagerオブジェクトでフォーカスを移動/取得する

InteractiveObjectSampleE.fla

```
import fl.managers.FocusManager;

var fc1:FocusManager;
var i:int;

//FocusManagerオブジェクトを生成
function initFc():void {
  fc1=new FocusManager(this);  ①
}
```

```
//tabIndexを設定
function setTabIndex():void {
  button1.tabIndex=1;
  button2.tabIndex=2;
  button3.tabIndex=3;
  button4.tabIndex=4;
}

//MovieClipにtabEnabled, focusRect, イベントハンドラを設定
function setEventHandler():void {
  for (i = 1; i <= 4; i++) {
    this["button"+i].addEventListener( FocusEvent.FOCUS_IN, setFcs );
    this["button"+i].addEventListener( FocusEvent.FOCUS_OUT, setFcs
    );
  }
}

//tabEnabedプロパティの設定
function setTabEnabledSettings( enableTab:Boolean ):void {
  for (i = 1; i <= 4; i++) {
    this["button"+i].tabEnabled=enableTab;
  }
}

//focusRectプロパティの設定
function setFocusRectSettings( enableRect:Boolean ):void {
  for (i = 1; i <= 4; i++) {
    this["button"+i].focusRect=enableRect;
  }
}

//focusIn / focusOutのイベントハンドラ
function setFcs( event:FocusEvent ):void {
  switch ( event.type ) {
    case FocusEvent.FOCUS_IN :
      event.target.alpha=0.2;
      //FocusManagerオブジェクトのメソッドを使用して現在のフォーカスオブジェクトを取得
      getFocus_txt.text=fc1.getFocus().name;                                  ②
      //FocusManagerオブジェクトのメソッドを使用して次のフォーカスオブジェクトを取得
```

```
        nextTabIndex_txt.text = fc1.getNextFocusManagerComponent().
        name;                                                          ②
        break;
      case FocusEvent.FOCUS_OUT :
        event.target.alpha=1.0;
        break;
    }
}

//初期呼び出し
initFc();
setEventHandler();
setTabIndex();
setTabEnabledSettings( true );
setFocusRectSettings( true );

//button1にフォーカスを移します
fc1.setFocus( button1 );                                               ③
```

7-3-5 「Tabによるフォーカス移動を制御する」のサンプルを修正してFocusManagerを使用します。6つのムービークリップのうち、右側の4つに対して、タブでフォーカス移動するサンプルです。①でFocusManagerオブジェクトを作成していますが、コンストラクタに管理対象とするDisplayObjectContainerオブジェクトを渡す必要があります。画面下部にgetFocus()メソッドによる現在フォーカスを持つオブジェクト、getNextFocusManagerComponent()メソッドにより、次にフォーカスを持つオブジェクトをそれぞれ取得してnameプロパティを表示します。これは②のfocusInイベントハンドラで行っています。③でsetFocus()メソッドを使用してフォーカスをbutton1に移しています。このサンプルをムービープレビューするときは、[制御]-[キーボードショートカットを無効]メニューにチェックをつけます。

4
1

7-4-2 Enterキーに反応するデフォルトボタンを作成する

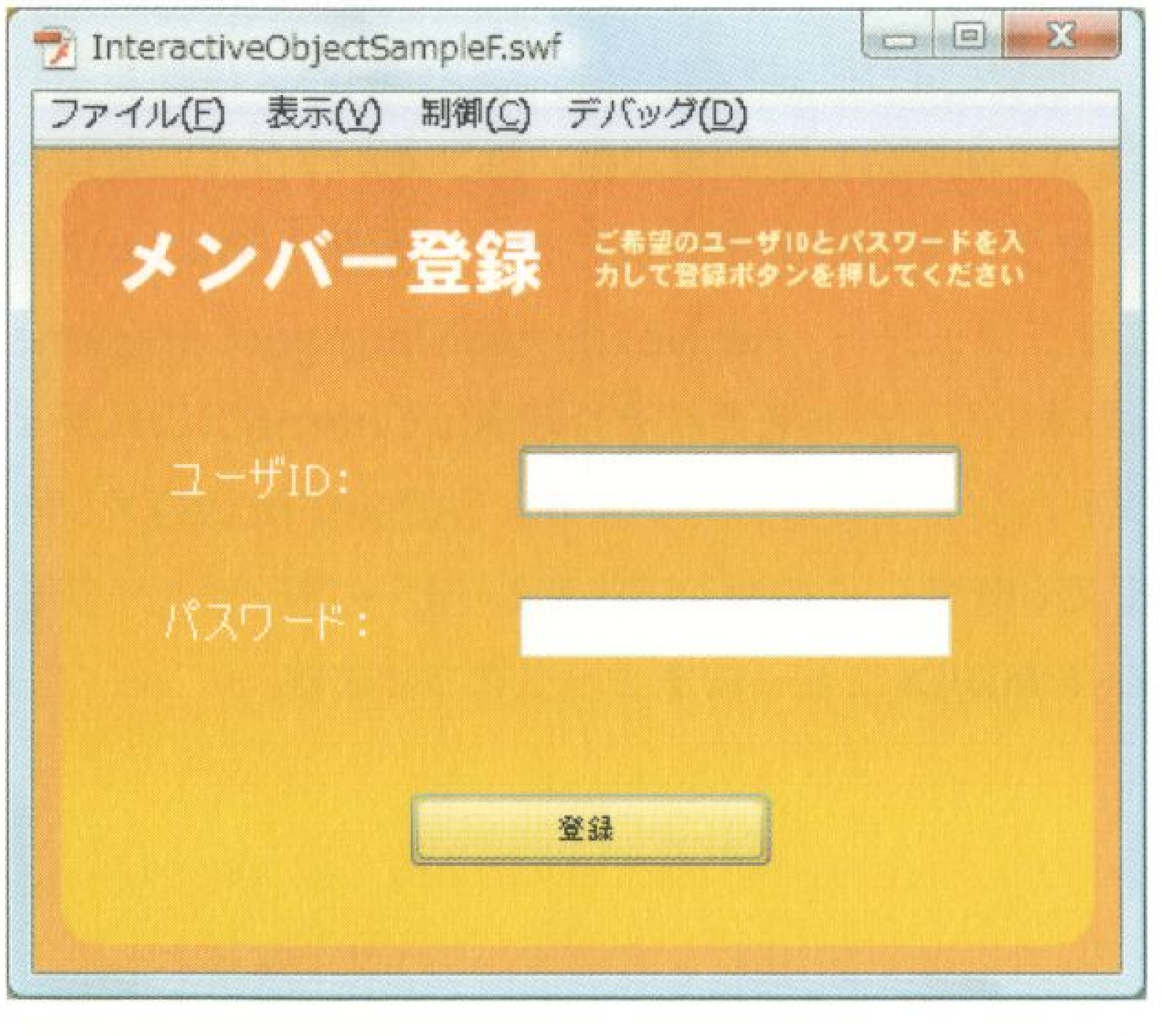
PART
1
2
3
4
5
6
7
8
9

入力フォームなどで、Enterキーを押したときに送信ボタンが押されるような処理を行いたい場合、デフォルトボタンを設定します。FocusManagerクラスにはデフォルトボタンを設定するためのdefaultButtonプロパティがあります。

FocusManagerクラスのデフォルトボタンに関するプロパティ詳細

プロパティ	プロパティの詳細
defaultButton :Button	FocusManagerインスタンスが管理するタブグループ内でデフォルトボタンを設定/取得します。FocusManagerオブジェクトは、Enterキーが押されたとき、フォーカスを持っているコンポーネントではなく、このプロパティで設定されたボタンに対してclickイベントを送出します。この処理はフォーカスがButtonコンポーネントにあるときは行われません。
defaultButton Enabled:Button	デフォルトボタンを有効/無効にします。 **true** ：FocusManagerはEnterキーが押された場合、設定されたデフォルトボタンにclickイベントを送出します。 **false**：FocusManagerはEnterキーの入力を監視せず、デフォルトボタンにclickイベントを送出しません。

SAMPLE



SAMPLE CODE

デフォルトボタンの設定

InteractiveObjectSampleF.fla

```
import fl.managers.FocusManager;
```

```
var fc1:FocusManager;

//FocusManagerオブジェクトを生成し、デフォルトボタンのための設定を行います
function initFc():void {                                                    ┐
  fc1=new FocusManager(this);                                               │
  fc1.defaultButtonEnabled = true;                                          │
  fc1.defaultButton =this.default_button;                                   ├─①
  default_button.tabEnabled = false;                                        │
  default_button.addEventListener(MouseEvent.CLICK, default                 │
  ButtonClicked );                                                          │
}                                                                           ┘

//デフォルトボタンが押されたときの処理
function defaultButtonClicked( event:MouseEvent ):void {                    ┐
  trace( "登録ボタンが押されました。" );                                    ├─②
}                                                                           ┘

//tabIndexを設定
function setTabIndex():void {
  name_txt.tabIndex=1;
  password_txt.tabIndex=2;
}

//初期呼び出し
initFc();
setTabIndex();
fc1.setFocus( name_txt );
```


入力フォームのサンプルです。2つのTextInputコンポーネントを配置しており、①の関数でFocusManagerにより、登録ボタンをデフォルトボタンに設定しています。この設定により、TextInputにフォーカスがあるときにEnterキーを押すと、このボタンのclickイベントが発生します。サンプルでは、②のハンドラを登録しているので、trace()関数が実行されます。このサンプルをムービープレビューするときは、[制御]-[キーボードショートカットを無効]メニューにチェックをつけます。

7－5 MovieClip

7-5-1 MovieClipを使う

ActionScript1.0、2.0では、表示エレメントはすべてMovieClipクラスで管理していましたが、ActionScript3.0では、Flash Playerのステージに表示されるビジュアルエレメントはすべてDisplayObjectクラスを基本とするクラスで、MovieClipクラスはタイムラインを持つ表示オブジェクトコンテナとして位置づけられています。そのため、MovieClipクラスが持つ特有の機能としてはタイムラインに関する操作が中心になります。

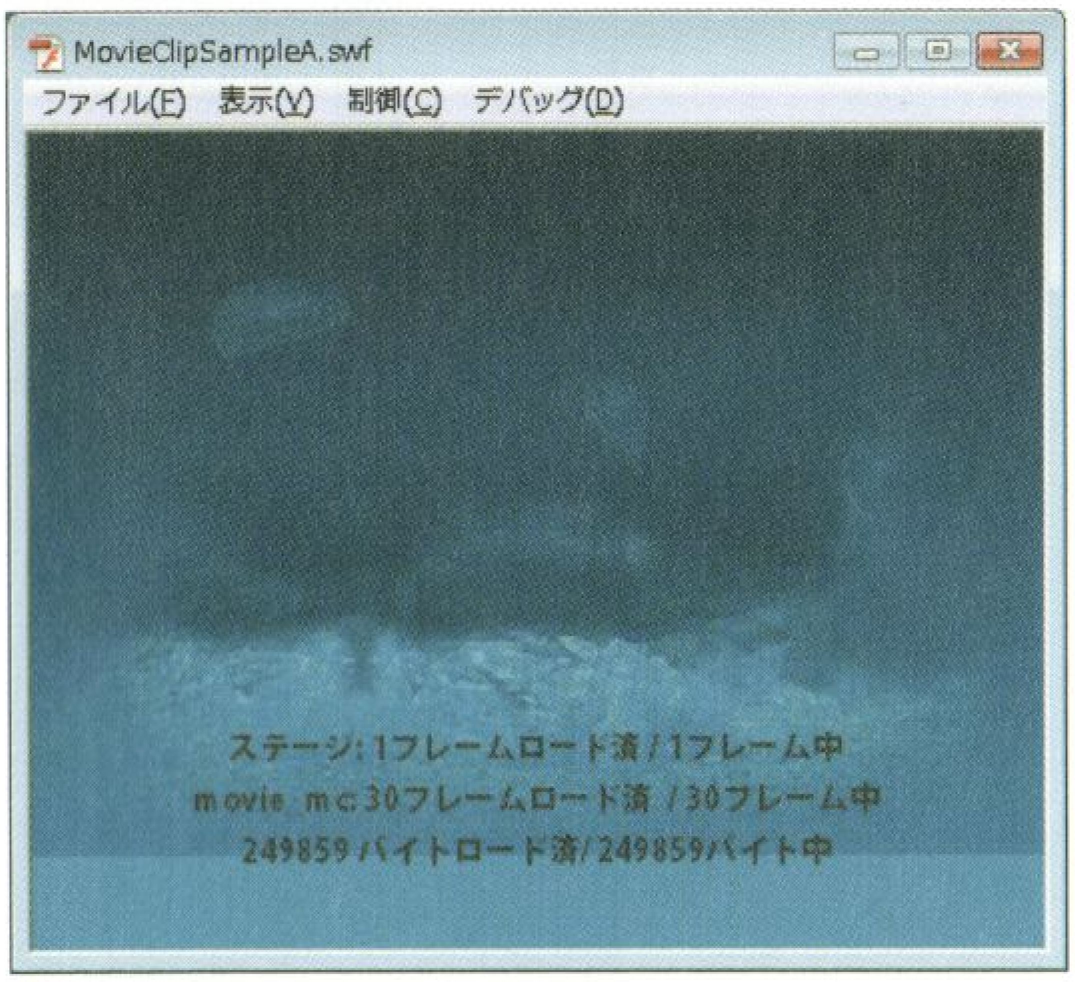
MovieClipクラスの主なプロパティ概要

目的	プロパティ名	プロパティの概要
再生ヘッドがあるフレームの情報を得る	currentFrame :int	現在再生ヘッドが位置するフレームの番号を表します。
	currentFrame Label:String currentLabel :String	現在再生ヘッドが位置するフレームのラベルを表します。
現在のシーンを得る	currentScene :Scene	現在再生ヘッドが置かれているシーンを得ることができます。
現在のシーンにあるラベルを得る	currentLabels :Array	現在のシーンに存在するラベルのリストをFrame Labelオブジェクトの配列で取得します。
ボタンタイプのムービークリップを有効/無効にする	enabled :Boolean	falseの場合、ムービークリップの「オーバー」、「ダウン」、「アップ」の各フレームは無効になります。
読み込み済みフレーム数を取得する	framesLoaded :int	ストリーミングSWFの読み込み済みフレーム数を取得できます。
LoaderInfoオブジェクトにアクセスする	loaderInfo :LoaderInfo	[DisplayObjectクラス] ファイルの読み込み情報を提供するLoaderInfoオブジェクトへのアクセスを提供します。
ムービークリップ内のすべてのシーンを取得する	scenes :Array	MovieClipインスタンス内のシーンの名前、フレーム数、フレームラベルがリストされた Sceneオブジェクトの配列を得ることができます。
ムービークリップ内のフレーム数を取得する	totalFrames :int	MovieClipインスタンス内のフレームの総数を得ることができます。framesLoadedプロパティと合わせて使用すると、何パーセントのフレームを読み込み済みかを判定できます。

MovieClipクラスの主なメソッド概要

目的	メソッド名	メソッドの概要
指定フレームから再生を開始する	gotoAndPlay(frame:Object, scene:String = null):void	フレーム番号またはフレームラベルを指定して再生を開始します。
指定フレームに再生ヘッドを移動する	gotoAndStop(frame:Object, scene:String = null):void	フレーム番号またはフレームラベルを指定して、そこに再生ヘッドを移動して停止します。
次/前のフレームに移動する	nextFrame():void prevFrame():void	次/前のフレームに再生ヘッドを移動してそこで停止します。
次/前のシーンに再生ヘッドを移動する	nextScene():void prevScene():void	次/前のシーンに再生ヘッドを移動します。
再生を開始/停止する	play():void stop():void	ムービークリップオブジェクトのタイムライン再生を開始/停止します。

SAMPLE



SAMPLE CODE

MovieClipオブジェクトの情報を調べる

MovieClipSampleA.fla

```
//loaderInfoを取得してイベントリスナーを登録
this.loaderInfo.addEventListener(ProgressEvent.PROGRESS,
progressHandler ); ―――――――――――――――――――――――――――― ①
function progressHandler(event:ProgressEvent):void {
  this.progress_txt.appendText("メインタイムライン: " +
  this.framesLoaded + "フレームロード済 / " + this.totalFrames +   ②
  "フレーム中¥n" );
```

```
    this.progress_txt.appendText( "movie_mc: " +
    this.movie_mc.framesLoaded + "フレームロード済 / " +
    this.movie_mc.totalFrames + "フレーム中¥n" );        ③
    this.progress_txt.appendText( this.loaderInfo.bytesLoaded +
    " バイトロード済/ " + this.loaderInfo.bytesTotal + "バイト中¥n" );   ④
}
```

①の「this.loaderInfo」という記述でこのムービーのロード情報を持つLoaderInfoオブジェクトを取得できます。このオブジェクトは読み込み状況を更新するためのprogressイベントを発生します。そのハンドラで現在の読み込み状況を把握できます。②ではメインタイムラインのフレーム数（totalFrames）と、ロード済みフレーム数（framesLoaded）を取得し、③ではmovie_mcというムービークリップのフレーム数を取得しています。また、ファイルの総バイト数とロード済みバイト数はbytesTotal、bytesLoadedプロパティで取得できますが、④でそれらを使用しています。

7-5-2 MovieClipの再生と停止を行う

タイムラインの再生と停止、フレームの移動などの操作はMovieClipクラスのメソッドで行います。複数のシーンを利用している場合、シーン名を指定して制御を行うことができます。

MovieClipクラスのフォーカスの移動/取得に関するメソッド詳細

メソッド	メソッドの詳細
gotoAndPlay(frame:Object, scene:String=null):InteractiveObject	**frame**：このパラメータで指定したフレーム番号/ラベル文字列で再生を開始します。注意として、現在のフレームに未実行のアクションがある場合、それらが実行された後にフレームの移動が行われます。 **scene**：シーンを指定して再生を制御する場合、シーン名をこのパラメータに設定します。現在のシーンで実行する場合、省略可能です。
gotoAndStop(frame:Object, scene:String=null):InteractiveObject	**frame**：このパラメータで指定したフレーム番号/ラベル文字列に移動して再生を停止します。現在のフレームに未実行のアクションがある場合、それらが実行された後にフレームの移動が行われます。 **scene**：シーンを指定する場合、シーン名をこのパラメータに設定します。現在のシーンで実行する場合、省略可能です。
nextFrame():void	次のフレームに移動して再生を停止します。現在のフレームに未実行のアクションがある場合、それらが実行された後にフレームの移動が行われます。
play():void	再生ヘッドを再生状態にして次のフレームに移動します。
prevFrame():void	前のフレームに移動して再生を停止します。現在のフレームに未実行のアクションがある場合、それらが実行された後にフレームの移動が行われます。
prevScene():void	前のシーンに移動します。現在のフレームに未実行のアクションがある場合、それらが実行された後にフレームの移動が行われます。
stop():void	再生ヘッドを停止状態にして次のフレームに移動しないようにします。

SAMPLE



SAMPLE CODE

再生ヘッドの制御 MovieClipSampleB.fla

```
//再生、停止ボタンのイベントハンドラ登録
play_btn.addEventListener( MouseEvent.CLICK, gotoAndPlay_ );  ①
stop_btn.addEventListener( MouseEvent.CLICK, gotoAndStop_ );

//gotoAndPlayを行う関数
function gotoAndPlay_( event:Event ):void {
  movie_mc.gotoAndPlay( play_ns.value );  ②
}

//gotoAndStopを行う関数
function gotoAndStop_( event:Event ):void {
  movie_mc.gotoAndStop( stop_ns.value );  ③
}
```

①はボタンハンドラの登録です。②ではmovie_mcに対して、NumericStepperの値をフレーム番号として、gotoAndPlay()メソッドで指定フレームから再生を行います。③では、同様にgotoAndStop()メソッドにより、フレームに移動して再生を停止します。

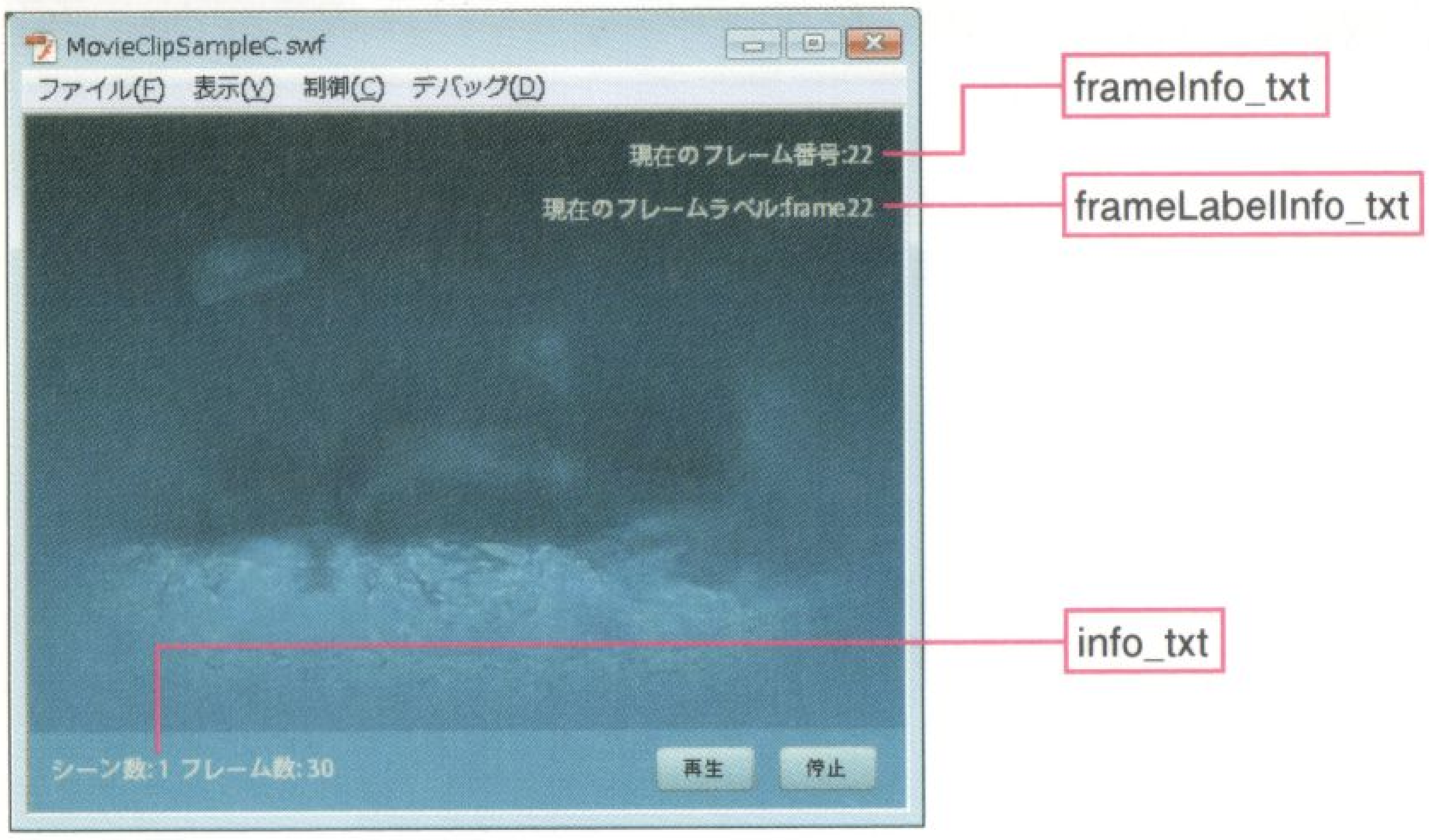
7-5-3 MovieClipのフレームとシーン情報を得る

MovieClipオブジェクトが持つフレームとシーンの情報は、MovieClipクラスのプロパティで調べます。現在どのフレームにいるかという情報も得られます。

MovieClipクラスのフレームとシーン情報に関するプロパティ詳細

<table>
<tr><th>プロパティ</th><th>プロパティの概要</th></tr>
<tr><td>currentFrame
:int</td><td>再生ヘッド位置の現在のシーン内でのフレーム番号を表します。他のシーンのフレーム数は含まれません。</td></tr>
<tr><td>currentFrameLabel
:String</td><td>現在のフレームに設定されているラベルを返します。ラベルがない場合、nullになります。</td></tr>
<tr><td>currentLabel
:String</td><td>現在のフレームに設定されているラベルを返します。ラベルがない場合、前のフレームにラベルがあればそのラベルの値が引き継がれます。前のフレームにもラベルがない場合はnullになります。</td></tr>
<tr><td>currentLabels
:Array</td><td>現在のシーンが持つすべてのフレームラベルを配列で取得できます。各ラベルの情報はFrameLabelクラスのインスタンスとして保持されます。FrameLabelクラスは次の情報を持ちます。
flash.display.FrameLabelクラスのプロパティ
<table>
<tr><th>プロパティ</th><th>内容</th></tr>
<tr><td>frame:int</td><td>nameプロパティで示されるラベルが設定されたフレームの番号</td></tr>
<tr><td>name:String</td><td>ラベルとして設定された文字列</td></tr>
</table></td></tr>
<tr><td>currentScene
:Scene</td><td>現在再生ヘッドがあるタイムラインが属しているシーン名を得ることができます。</td></tr>
<tr><td>scenes
:Array</td><td>MovieClipインスタンス内のシーンの名前、フレーム数、フレームラベルが格納されたSceneオブジェクトを配列で返します。Sceneクラスは次の情報を持ちます。
flash.display.Sceneクラスのプロパティ
<table>
<tr><th>プロパティ</th><th>内容</th></tr>
<tr><td>labels
:Array</td><td>シーンが持つすべてのフレームラベルを配列で取得できます。各ラベルの情報はflash.display.FrameLabelクラスのインスタンスとして保持されます。</td></tr>
<tr><td>name
:String</td><td>シーン名を表します。</td></tr>
<tr><td>numFrames:
int</td><td>シーン内のフレーム数を表します。</td></tr>
</table></td></tr>
<tr><td>totalFrames
:int</td><td>MovieClipインスタンス内のフレームの総数を表します。インスタンス内に複数のフレームがある場合、このプロパティは、ムービークリップ内のすべてのシーンのフレーム総数を表します。</td></tr>
</table>

SAMPLE



SAMPLE CODE

現在のフレーム、シーン情報を調べる

MovieClipSampleC.fla

```
//再生、停止ボタンのイベントハンドラ登録
play_btn.addEventListener( MouseEvent.CLICK, Play_ );
stop_btn.addEventListener( MouseEvent.CLICK, Stop_ );
//Playを行う関数
function Play_( event:Event ):void {
  movie_mc.play();
}
//Stopを行う関数
function Stop_( event:Event ):void {
  movie_mc.stop();
}
movie_mc.addEventListener( Event.ENTER_FRAME, showInfo );
function showInfo(event:Event):void {
  franeInfo_txt.text = "現在のフレーム番号:" + movie_mc.currentFrame ;
  franeLabelInfo_txt.text = "現在のフレームラベル:" + movie_mc.
  currentFrameLabel ;
}
info_txt.text = "シーン数: " + movie_mc.scenes.length + " フレーム数:
" + movie_mc.totalFrames;
```

①play()メソッドとstop()メソッドを呼び出すためのイベントハンドラの登録

②フレーム番号とフレームラベルを取得してテキストフィールドに書き出す

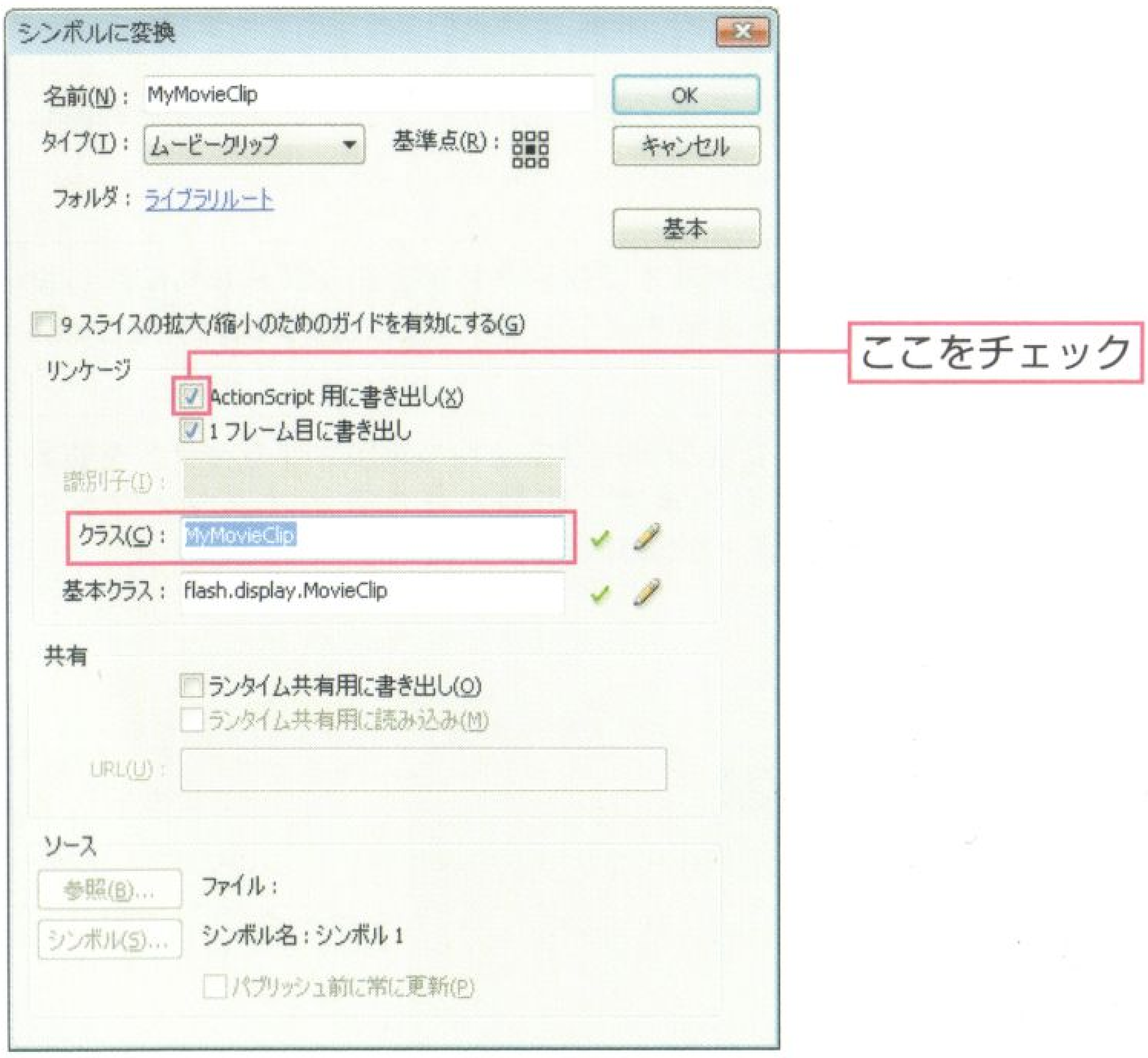
③シーン数とフレーム数を調べる

このサンプルに配置されたコンテンツはビデオを30フレームのフレームベースのアニメーションとして再生するもので、MovieClipの名前はmovie_mc、その各フレームにframe X (Xは1-30の数値) という名前がつけられています。

7-5-4 ライブラリシンボルのインスタンスをステージに配置する

ライブラリ内のMovieClipシンボルのインスタンスをステージに配置する場合、DisplayObjectContainerクラスのaddChild()メソッドで行うことができます。インスタンスの生成はnew演算子で行いますが、そのときに使用するクラス名はシンボルを作成したときにつけたシンボル名になります。また、シンボルに変換するときに「ActionScript用に書き出し」オプションを指定しておく必要があります。

■ シンボルに変換ダイアログ



ダイアログで[詳細]ボタンを押して、「ActionScript用に書き出し」にチェックをつけてシンボルに変換します。

■ DisplayObjectContainerクラスの子オブジェクト追加/削除に関するプロパティ詳細

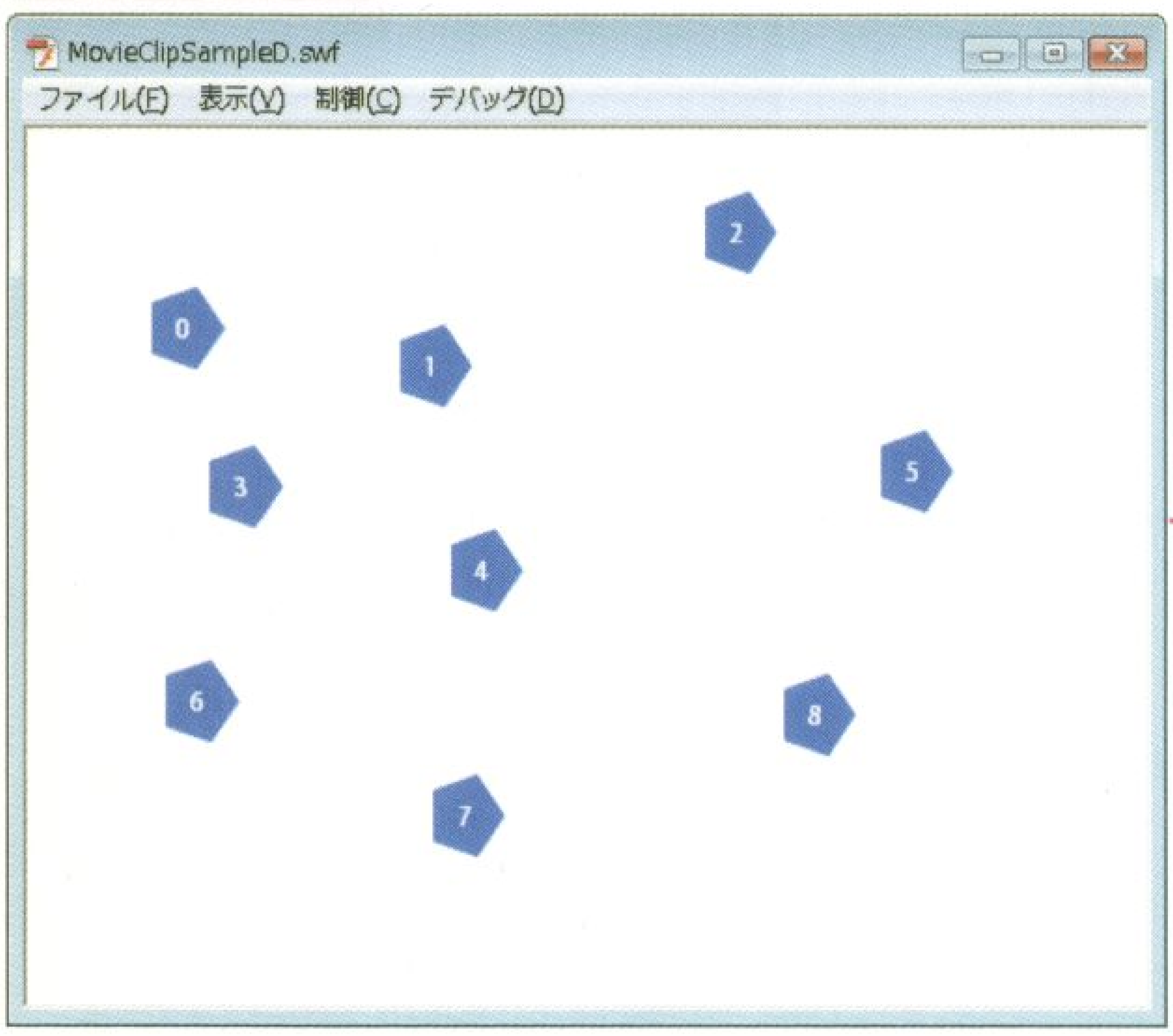
プロパティ	プロパティの詳細
numChildren: int	DisplayObjectContainerオブジェクトの子の数を表します。読み取り専用で、値を設定することはできません。このプロパティの値はaddChild()、removeChild()などのメソッドで変化します。

PART
1
2
3
4
5
6
7
8
9

DisplayObjectContainerクラスの子オブジェクトの追加/削除に関するメソッド詳細

メソッド	メソッドの詳細
addChild (child:DisplayObject) :DisplayObject	childをDisplayObjectContainerオブジェクトの子として追加します。追加された表示オブジェクトは既存の子オブジェクトの上に配置され、前面に表示されます。childが他のDisplayObjectContainerオブジェクトの子であった場合、その子オブジェクトは削除されます。
addChildAt (child:DisplayObjact, index:int) :DisplayObject	childをDisplayObjectContainerオブジェクトの子として追加します。indexにより、位置を指定して追加できます。位置は0が最背面で、1,2,...の順に前面に追加されます。すでに使用されている番号を指定した場合、その番号以上の子オブジェクトのインデックスは繰り上げられます。childが他のDisplayObjectContainerオブジェクトの子であった場合、その子オブジェクトは削除されます。
getChildIndex (child:DisplayObjact) :int	childで指定した子オブジェクトのインデックス位置を取得します。
removeChild (child:DisplayObject) :DisplayObject	childで指定した表示オブジェクトを表示リストから削除します。削除された表示オブジェクトのparentプロパティがnullに設定されます。
removeChildAt (index:int):DisplayObject	indexで指定したインデックスを持つ表示オブジェクトを表示リストから削除します。削除された表示オブジェクトのparentプロパティがnullに設定されます。

SAMPLE



マウスボタンを9回クリックして
インスタンスを9個作成したところ

SAMPLE CODE

ライブラリシンボルのインスタンスをステージに配置/削除する

MovieClipSampleD.fla

```
//ステージ上でマウスをクリックしたときにライブラリシンボルをインスタンス化する関数を呼びます
this.stage.addEventListener( MouseEvent.CLICK, attachMovie_ );
//ライブラリシンボルをインスタンス化する関数
function attachMovie_( event:MouseEvent ):void {
  //ctrlキーが押されていたら、最後のインデックスを持つ子を削除
  if (event.ctrlKey) {
    if ( this.numChildren > 0 )
    this.removeChildAt( this.numChildren - 1 );      ①
    return;
  }
  //altキーが押されていたら、先頭のインデックスを持つ子を削除
  if (event.altKey) {
    if ( this.numChildren > 0 )
    this.removeChildAt( 0 );                         ②
    return;
  }
  //ここからライブラリシンボルをインスタンス化して配置する手順
  var obj = this.addChild( new symbol1() );
  obj.x=event.stageX;                                ③
  obj.y=event.stageY;
}
```


このサンプルでは、ActionScript用に書き出し指定されたシンボル(symbol1)をステージ上に配置します。配置はマウスのクリックで行い、マウスポインタの場所にインスタンスを移動します。①でライブラリシンボルをインスタンス化して配置しています。Ctrlキーを押しながらクリックした場合、いちばん大きいインデックス値を持つ子を削除します。これは①で行っています。②では、altキーを押しながらクリックした場合、先頭(0)のインデックスを持つ子を削除します。

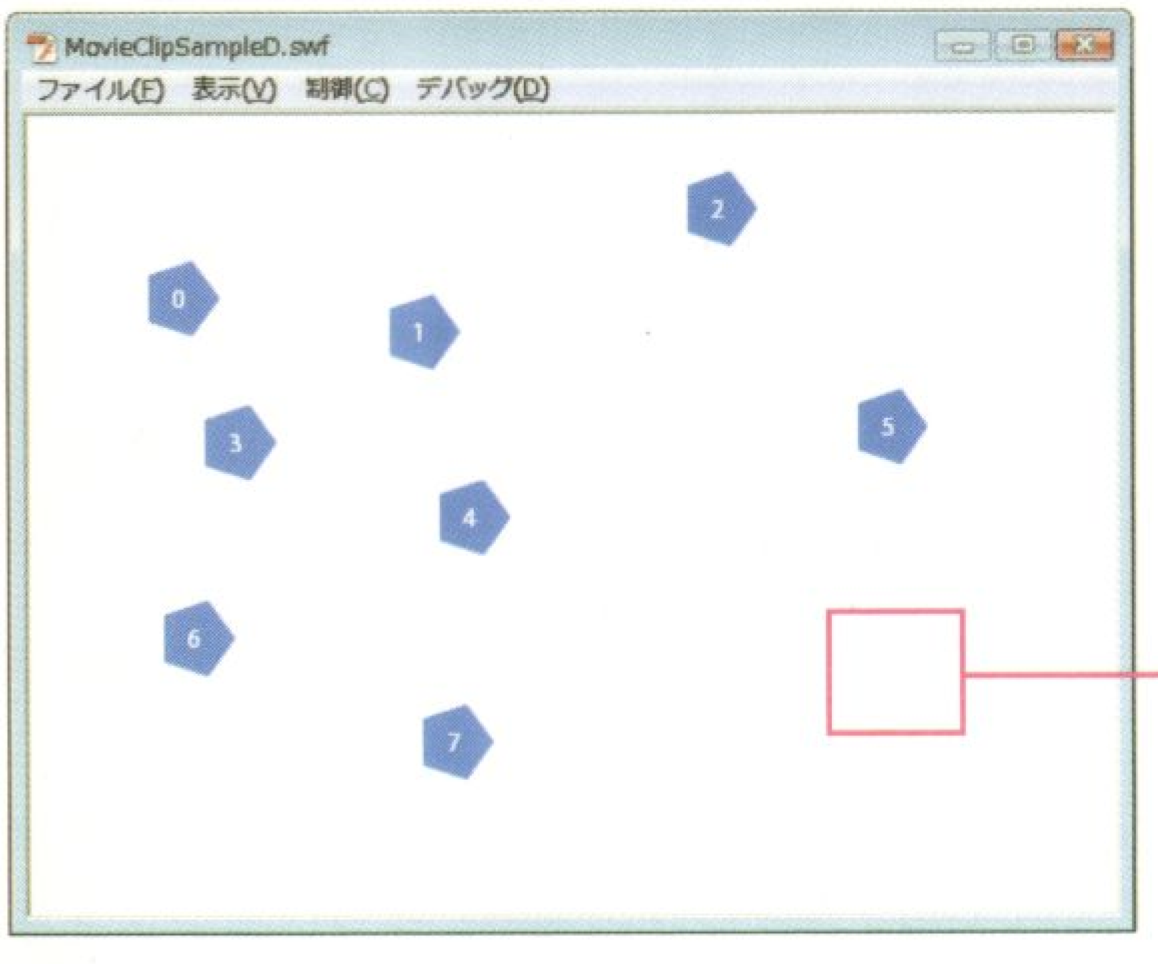
SAMPLE CODE

symbol1 シンボルのフレームアクション

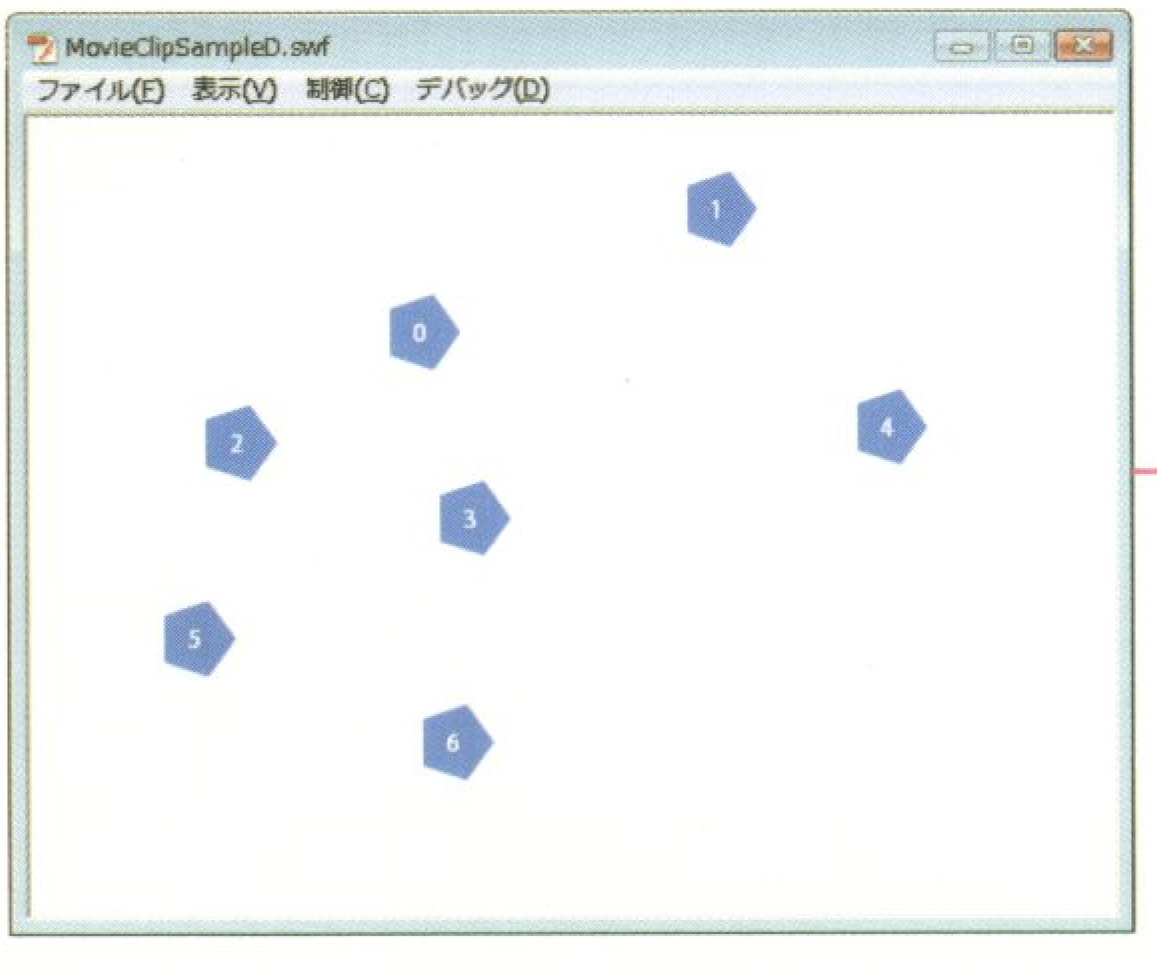
```
//showChildIndex関数をチェックするためのイベントハンドラ登録
this.addEventListener( Event.ENTER_FRAME, showChildIndex );
//ChildIndexをチェックするための関数
function showChildIndex( event:Event ):void {
```

```
  //parentがnullの場合、すでにChildIndexを持たないため、ハンドラを削除して終了
  if ( parent == null ) {
    this.removeEventListener( Event.ENTER_FRAME, showChildIndex );
    return;
  }
  //現在のChildIndex値をテキストフィールドに記入
  text_txt.text = parent.getChildIndex( this ).toString();
}
```


シンボルのタイムラインアクションとして、自分の子Index番号を表示する処理を行っています。Index値は変化するため、enterFrameのイベントで逐次判定を行うようにしています。子が削除されるとparentプロパティがnullになるため、その場合イベントの処理を終了するようにしています。



Ctrlキーを押しながらマウスをクリックして9番目の子が削除されたところ



altキーを押して先頭の子を削除したところ。インデックス値が繰り下がっている

7-6-1 Loaderクラスを使う

ActionScript3.0では、外部SWFファイルを現在のムービーに読み込む場合Loaderクラスを使用します。Loaderクラスのload()メソッドでロードを開始し、中止する場合はcancel()メソッド、ロード後に削除する場合はunload()メソッドを使用します。LoaderクラスはJPG、PNG、GIFなどのイメージファイルのロードにも使用されます。この章ではSWFファイルのロードを中心に解説します。Loaderクラスによる外部ファイルロードの基本手順は次のようになります。FlashPlayerのセキュリティ制限により、SWFの発行元ドメインとロードするファイルのドメインは同一である必要があります。

Loaderクラスによる外部ファイルロードの基本手順

順番	処理内容
1	ロードするファイルを示す指定するためのflash.net.URLRequestクラスのインスタンスを生成します。そのとき、コンストラクタにロードするファイル名を指定します。パスはSWFを基準とする相対パス、http:// で始まるURLのどちらも指定できます。
2	flash.display.Loaderクラスのインスタンスを生成します。
3	Loaderオブジェクトでload()メソッドを実行します。そのとき、パラメータに1．で作成したURLRequestオブジェクトを指定します。
4	ロードしたコンテンツを配置するDisplayObjectContainerオブジェクトでaddChild()メソッドを実行します。そのとき、パラメータに2．で作成したLoaderインスタンスを指定します。ロードされたファイルはLoaderオブジェクトの子として追加されます。

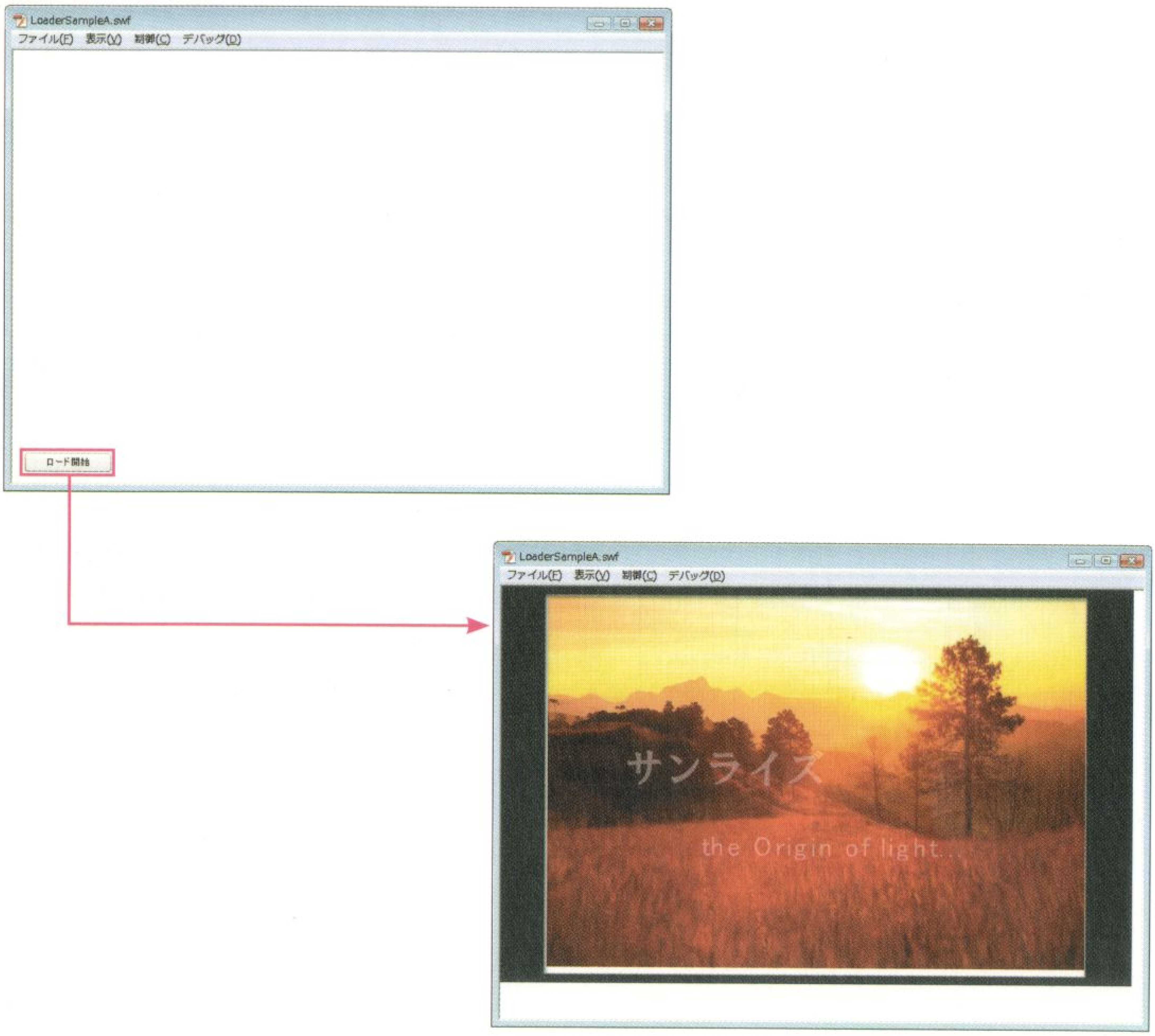
Loaderクラスの主なプロパティ概要

目的	プロパティ名	プロパティの概要
読み込んだファイルにアクセスする	content :DisplayObject	load()メソッドでロードされたファイルのルート表示オブジェクトへの参照を得ることができます。
ファイルの読み込み状況と、ファイルに関する大きさなどの情報を得る	contentLoaderInfo :LoaderInfo	読み込みの状況と読み込んだファイルの規格サイズ、ActionScriptバージョンなどの情報を持つLoaderInfoインスタンスにアクセスできます。

Loaderクラスの主なメソッド概要

目的	メソッド名	メソッドの概要
Loaderオブジェクトを作成する	`Loader()`	コンストラクタは引数をとりません。読み込むファイル名はload()メソッドで指定します。
ファイル名を指定して読み込みを開始する	`load( request:URLRequest, context:LoaderContext = null ):void`	SWF、イメージファイルの読み込みを開始します。
読み込み処理をキャンセルする	`close():void`	load()メソッドで開始した読み込みを中止します。
読み込まれたオブジェクトを削除する	`unload():void`	load()メソッドで読み込んだオブジェクトを削除します。
読み込まれたSWFの実行を中止して削除する	`unloadAndStop( gc:Boolean = true ):void`	読み込んだオブジェクトの実行を中止してオブジェクトの解放を試みます。

SAMPLE



サンプルを開始すると表示される「ロード開始」ボタンを押して、SWFのロードを開始します。

SAMPLE CODE

LoaderクラスでSWFファイルをロードする基本手順 LoaderSampleA.fla

```
import flash.net.URLRequest;
import flash.display.Loader;

var url:URLRequest;
var ldr:Loader;

//ロード開始ボタンのイベントハンドラ登録
this.startLoading_btn.addEventListener(MouseEvent.CLICK, startLoading
);

// SWFをロードする関数
function startLoading( event:MouseEvent ): void {
    // 1. URLRequestオブジェクトの生成                         ①
    url = new URLRequest( "sampleSWF01.swf" );
    // 2. Loaderオブジェクトの生成
    ldr = new Loader();
    // 3. load()メソッドの実行                                 ②
    ldr.load( url );
    // 4. addChild()メソッドの実行 ――― ③
    this.addChild( ldr );
    //ボタンを非表示に(再読み込み防止)
    this.startLoading_btn.visible = false;
}
```

「Loaderクラスによる外部ファイルロードの基本手順」で解説した手順のとおりにLoaderオブジェクトを作成してファイルをロードします。①の範囲がその手順です。②で指定している"sampleSWF01.swf"ファイルはこのサンプルのSWFファイルと同じ場所にあることを想定しています。③でaddChild()メソッドを実行していますが、このメソッドを実行しないと、ロードのみして画面には表示されません。

7-6-2 外部SWFファイルをロードする

PART
1
2
3
4
5
6
7
8
9

Loaderクラスを使用して外部SWFファイルをロードするときに重要となるのがロードの状況に基づくイベントになります。load()メソッドの実行に関連するイベントを把握しておく必要があります。ここでは、前節で解説した基本手順にイベントによる状況の取得を追加する方法を解説します。Loaderクラスが送出する主なイベントは下表のとおりですが、読み込みの状況を管理するのがLoarderInfoオブジェクトであるため、イベントもこのオブジェクトが送出します。そのため、イベントをリスンするオブジェクトはLoaderではなく、LoaderオブジェクトのcontentLoaderInfoプロパティで参照できるLoaderInfoオブジェクトになります。イベント処理は、イメージファイルをロードする場合も同様に行うことができます。

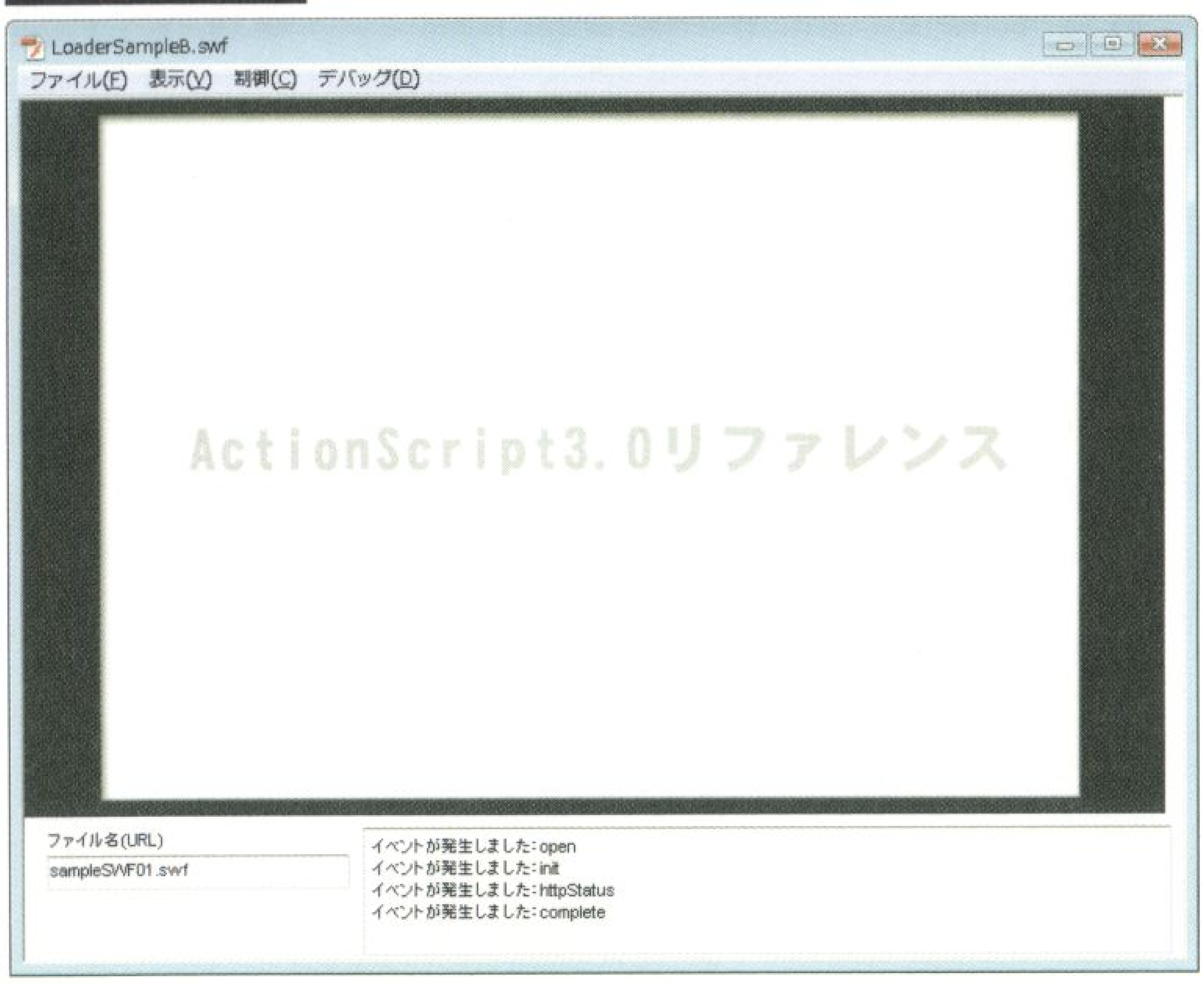
Loaderクラスのload()メソッドの実行で送出されるイベント詳細

イベント:クラス	内容
complete:Event	ファイルの読み込みが完了したときに、contentLoaderInfoオブジェクトによって送出されます。initイベントの後に送出されます。
httpStatus:HTTPStatusEvent	ネットワーク要求がHTTP上で行われ、Flash PlayerがHTTP状況コードを検出できる場合に、contentLoaderInfoオブジェクトによって送出されます。
init:Event	読み込まれたSWFファイルのプロパティおよびメソッドにアクセスできる場合に送出されます。initイベントはcompleteイベントの前に送出されます。initイベントの送出時は次の処理が完了しています。 読み込まれたオブジェクトに関係したすべてのプロパティとメソッド、およびLoaderInfoオブジェクトに関係したすべてのプロパティとメソッドにアクセス可能 すべての子オブジェクトのコンストラクタが完了済み ロードされたSWFのメインタイムラインの最初のフレームに含まれるすべてのActionScriptコードが実行済み
ioError:IOErrorEvent	入出力エラーが発生して読み込み処理が失敗したときに送出されます。
open:Event	読み込み処理が開始したときに送出されます。
progress:ProgressEvent	読み込み処理を実行中にデータを受信したときに、送出されます。
securityError:SecurityErrorEvent	ローカルファイルシステムのサンドボックス内のSWFファイルがネットワーク接続したローカルのサンドボックス内のコンテンツを読み込もうとしたか、またはその逆の場合に送出されます。
unload:Event	読み込まれたオブジェクトが削除されるときに送出されます。

Loaderクラスの読み込み情報に関するメソッド

メソッド	内容
load(request:URLRequest, context:LoaderContext = null):void	SWFファイル、イメージファイルをロードします。成功した場合、このLoaderオブジェクトの子として格納します。Loaderオブジェクトは1つの子しか格納できないため、同じオブジェクトで別のファイルをロードすると入れ替わりが発生します。読み込めるファイルは次のとおりです。 SWF JPEG プログレッシブJPEG 非アニメーションGIF PNG
close():void	現在進行中のload()メソッドの処理をキャンセルします。
unload():void	load()メソッドを使用して読み込まれたオブジェクトを、Loaderインスタンスの子から削除します。読み込まれたオブジェクト自体は可能な場合、削除されます。
unloadAndStop(gc:Boolean = true):void	子SWFファイルのアンロードを試み、ロードされたSWFファイルのコマンドの実行を中止します。子SWFでは次の処理が行われます。 サウンドは停止されます。 Stageのイベントリスナーが削除されます。 enterFrame、frameConstructed、exitFrame、activate、deactivateの各イベントリスナーが削除されます。 タイマーは停止されます。 CameraインスタンスおよびMicrophoneインスタンスが解放されます。 ムービークリップは停止されます。

SAMPLE



外部SWFファイルのロードが正常に完了したときにはこのような画面となります。

SAMPLE CODE

外部SWFをロードしてイベント処理を行う

LoaderSampleB.fla

```
import flash.net.URLRequest;
import flash.display.Loader;

var url:URLRequest;
var ldr:Loader;

//ロード開始ボタンのイベントハンドラ登録
this.startLoading_btn.addEventListener(MouseEvent.CLICK,
startLoading );

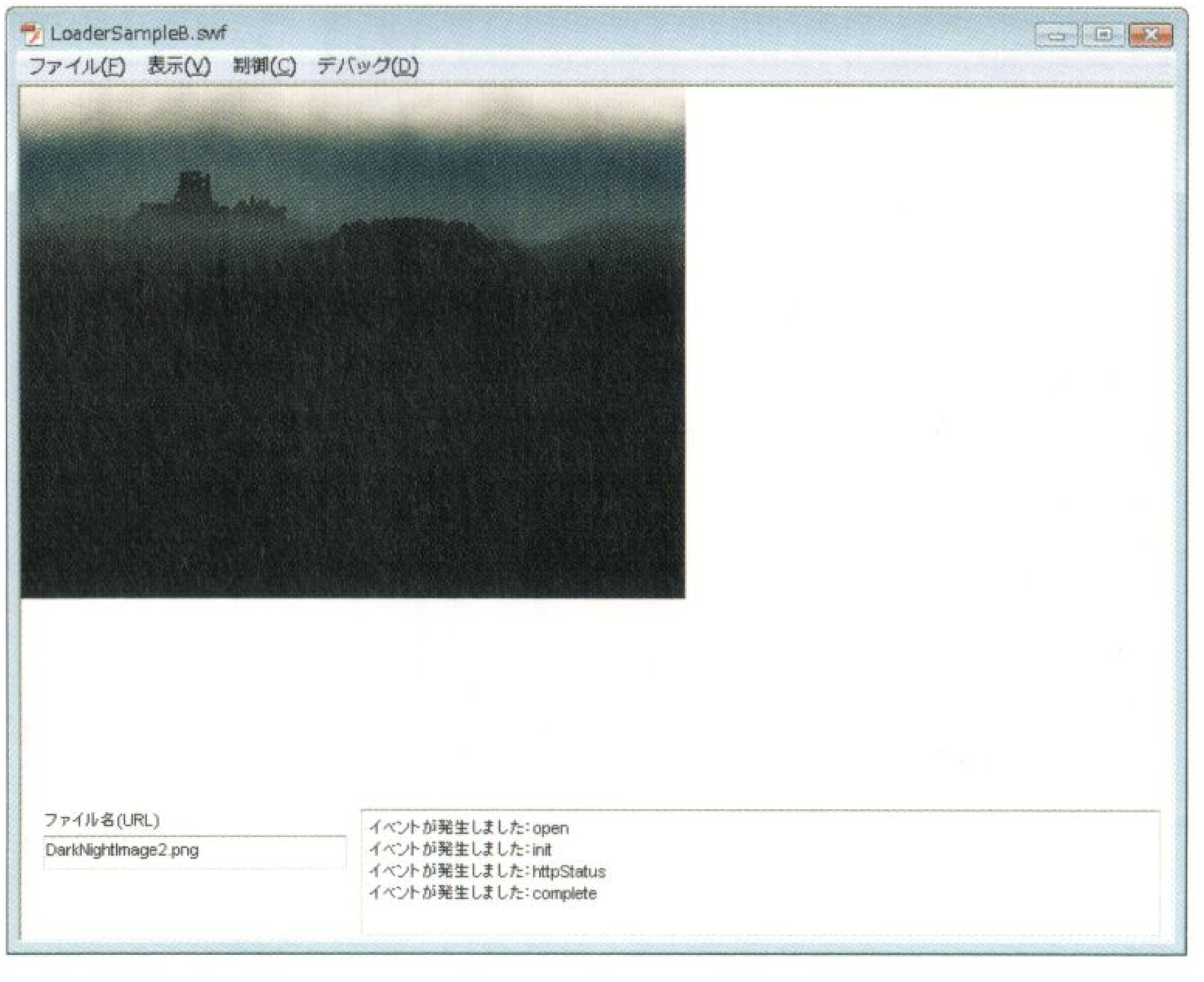
// SWFをロードする関数
function startLoading( event:MouseEvent ):void {
  // 1. URLRequestオブジェクトの生成
  url=new URLRequest( filename_txt.text );
  // 2. Loaderオブジェクトの生成
  ldr = new Loader();
  //Loaderオブジェクトのイベントハンドラを追加
  ldr.contentLoaderInfo.addEventListener(
      Event.COMPLETE, showEvent );
  ldr.contentLoaderInfo.addEventListener(
      HTTPStatusEvent.HTTP_STATUS, showEvent );
  ldr.contentLoaderInfo.addEventListener(
      Event.INIT, showEvent );                          ①
  ldr.contentLoaderInfo.addEventListener(
      Event.OPEN, showEvent );
  ldr.contentLoaderInfo.addEventListener(
      IOErrorEvent.IO_ERROR, showEvent );
  ldr.contentLoaderInfo.addEventListener( SecurityErrorEvent.
  SECURITY_ERROR, showEvent );
  // 3. load()メソッドの実行
  ldr.load( url );
  // 4. addChild()メソッドの実行
  this.addChild( ldr );
  //ボタンを非表示に(再読み込み防止)
```

```
  this.startLoading_btn.visible=false;
}

//イベントの発生を出力する関数
function showEvent(event:Event):void {
  output_txt.appendText( "イベントが発生しました:" + event.type + "¥n" );
}
```

Loaderオブジェクトによる基本手順は同じですが、load()メソッドを実行する前に、①で行っているように、イベントリスナーを登録しておきます。このサンプルでは、ロードの開始、完了、エラーなどの基本イベントを取得してテキストエリアに出力しています。SWFだけでなく、イメージファイルのロードも同様に行うことができます。ファイル名間違いなどで指定したファイルが存在しないとき、ioErrorが発生します。

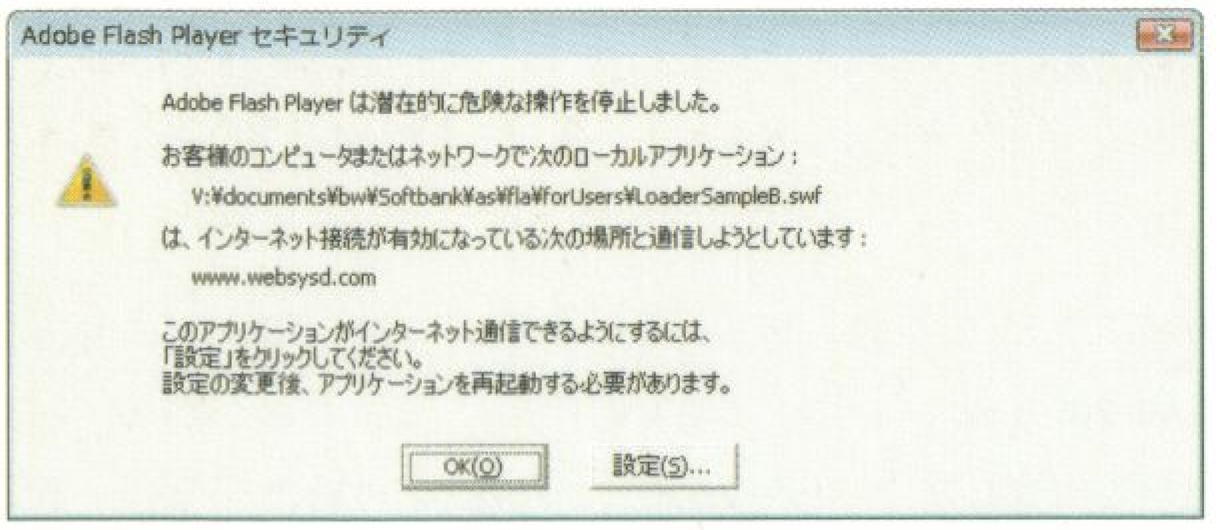
■ イメージファイル(PNG)をロードした場合



■ ロードエラー(ファイルが存在しない場合)

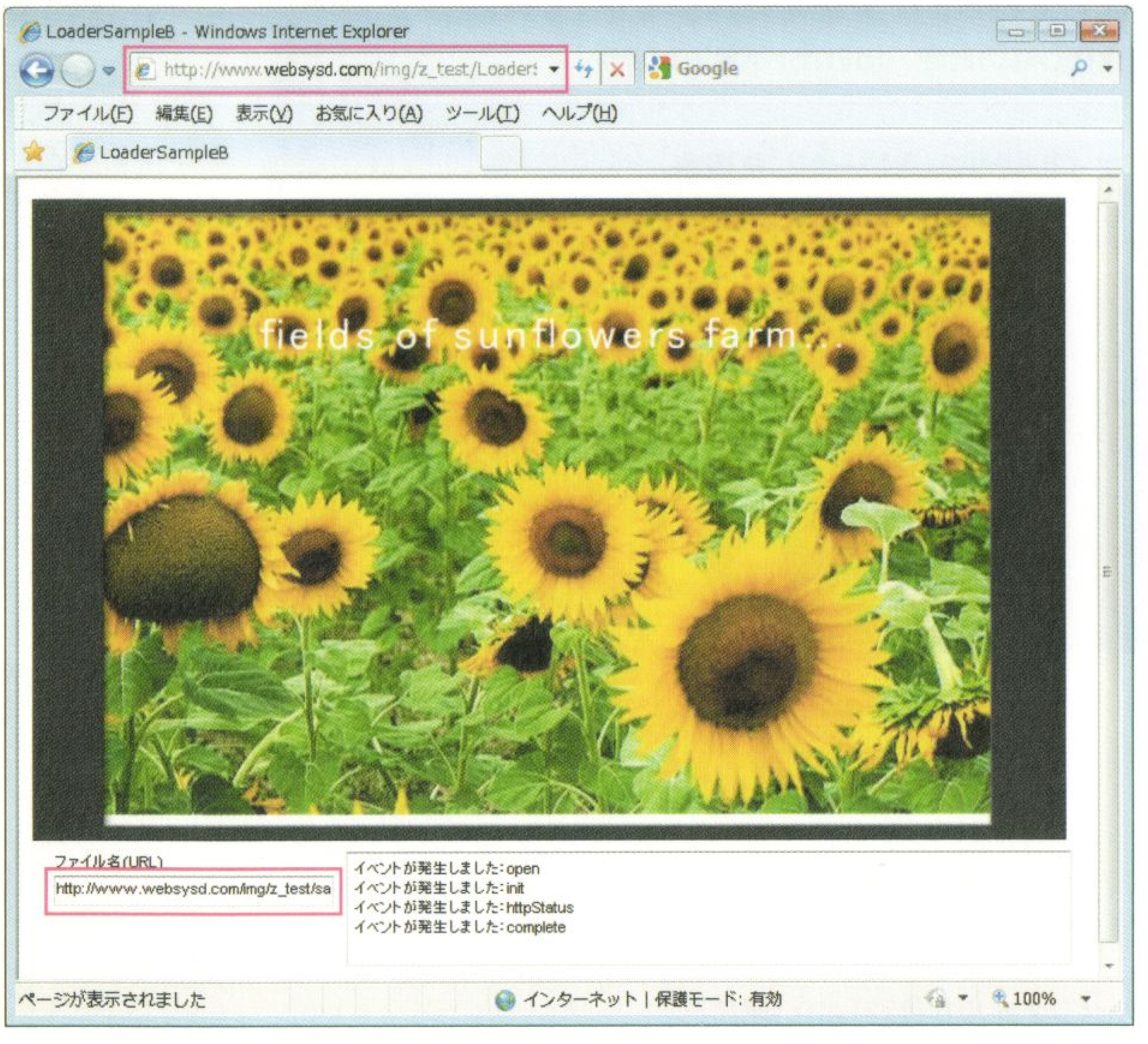


たとえばローカルファイル以外のファイル名(http://xxx.com/sampleSWF01.swfなど)を指定した場合、SecurityError: Error #2028が発生します。



このような場合、アプリケーションのSWFファイルとロードするファイルをWebサーバ（http://xxx.com）に配置して、読み込むファイルもhttp://xxx.com/sampleSWF01.swfのように指定すると、セキュリティエラーは表示されず、ファイルが正常にロードされます。

■ 正常にロードされた例



7-6-3 SWFファイルの読み込み情報を調べる

ロードするSWFファイルの読み込み状況をユーザに表示する必要があるとき、MovieClipクラスと、LoaderInfoクラスのプロパティとイベントが使用できます。MovieClipクラスのframesLoadedプロパティは読み込み済みのフレーム数を表します。このプロパティと全フレーム数を表すtotalFramesプロパティを計算すると全体の何%のフレームが読み込み済みかを調べることができます。読み込んだサイズの情報は、DisplayObjectクラスのloaderInfoプロパティで得ることができます。このプロパティは読み込みサイズに関するデータを保持するLoaderInfoクラスのインスタンスへのアクセスを提供します。LoaderInfoインスタンスが受けるprogressイベントで読み込み状況、completeイベントで読み込みの完了を知ることができます。

6
3

DisplayObjectクラスの読み込み情報に関するプロパティ詳細

プロパティ	プロパティの詳細
framesLoaded :int	SWFファイルのロード済みフレーム数を表します。
totalFrames :int	SWFファイルのトータルフレーム数を表します。
loaderInfo :LoaderInfo	SWFファイルの読み込み済みサイズ(バイト数)、全体サイズ(バイト数)を提供するLoaderInfoオブジェクトへの参照を提供します。このオブジェクトは読み込まれるSWF、イメージファイルに関する読み込み状況やサイズを提供します。

LoaderInfoクラスの読み込み情報に関するプロパティ詳細

プロパティ	プロパティの詳細
bytesLoaded :uint	SWFファイルの読み込み済みバイト数を表します。
bytesTotal :uint	SWFファイルのトータルバイト数を表します。

LoaderInfoクラスの読み込み情報に関するイベント詳細

イベント定数:下段は定数	イベントの詳細
progress	読み込み処理を実行中にデータを受信したときに、送出されます。
ProgressEvent.PROGRESS	

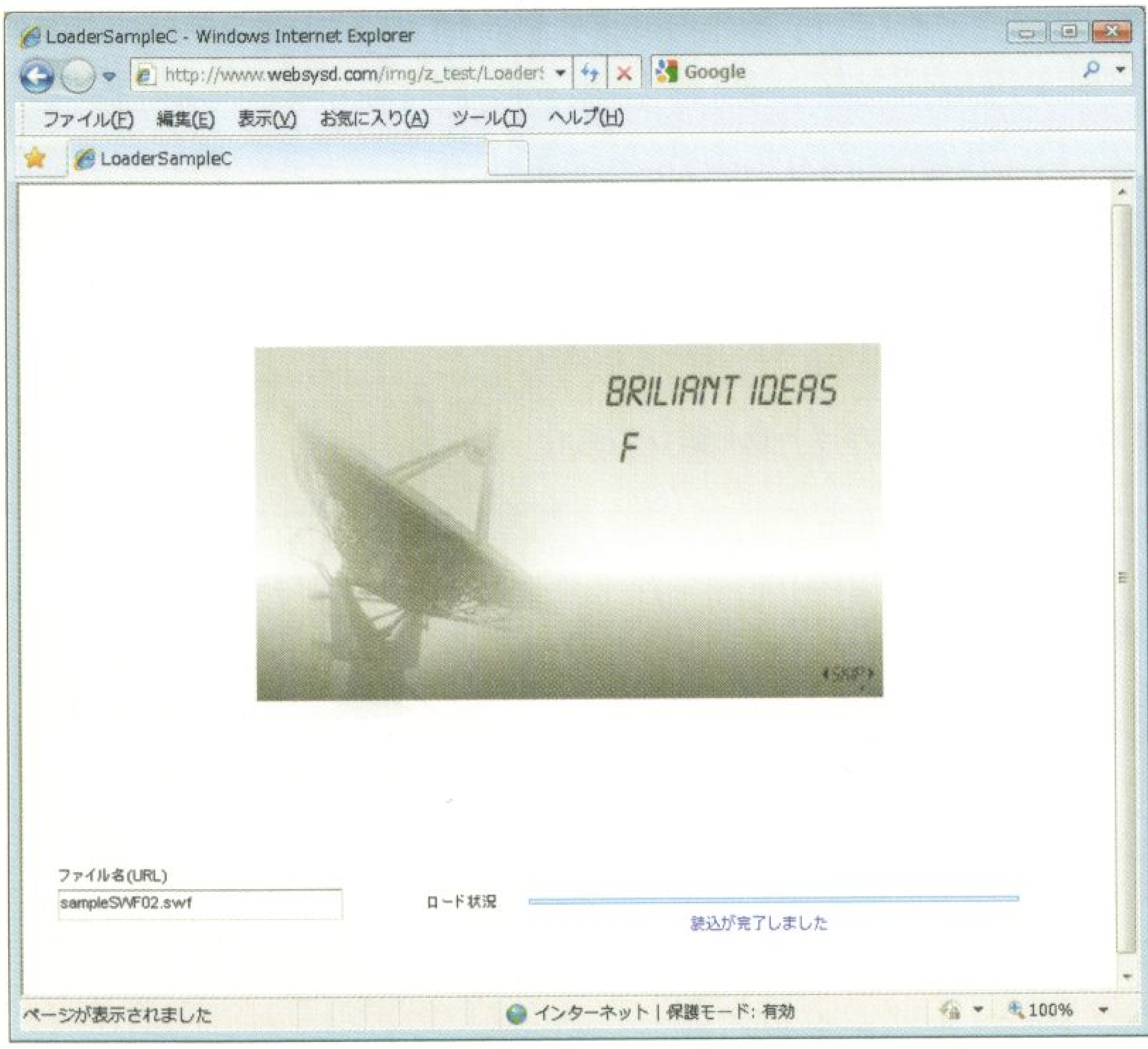
LoaderInfoクラスの読み込み情報に関するイベント詳細（続き）

イベント定数：下段は定数	イベントの詳細
init Event.INIT	読み込まれたSWFファイルのプロパティおよびメソッドにアクセスできる場合に送出されます。initイベントはcomplete イベントの前に送出されます。initイベントの送出時は次の処理が完了しています。 ・読み込まれたオブジェクトに関係したすべてのプロパティとメソッド、および LoaderInfoオブジェクトに関係したすべてのプロパティとメソッドにアクセス可能 ・すべての子オブジェクトのコンストラクタが完了済み ・ロードされたSWFのメインタイムラインの最初のフレームに含まれるすべてのActionScriptコードが実行済み
complete Event.COMPLETE	ファイルの読み込みが完了したときに、contentLoaderInfoオブジェクトによって送出されます。initイベントの後に送出されます。
ioError IOErrorEvent.IO_ERROR	入出力エラーにより読み込みが失敗したときに送出されます。

SAMPLE



ロード中を表すプログレスバー



ロードが完了して再生したSWF

SAMPLE CODE

SWFファイルの読み込み状況を調べる

LoaderSampleC.fla

```
import flash.net.URLRequest;
import flash.display.Loader;
import fl.controls.ProgressBarMode;
var url:URLRequest;
var ldr:Loader;
//ロード開始ボタンのイベントハンドラ登録
this.startLoading_btn.addEventListener(MouseEvent.CLICK,
startLoading );
// SWFをロードする関数
function startLoading( event:MouseEvent ):void {
  // 1. URLRequestオブジェクトの生成
  url=new URLRequest(filename_txt.text);
  // 2. Loaderオブジェクトの生成
  ldr = new Loader();
  //Loaderオブジェクトのイベントハンドラを追加
  ldr.contentLoaderInfo.addEventListener( ProgressEvent.
  PROGRESS, showEvent );
  ldr.contentLoaderInfo.addEventListener( Event.COMPLETE,
  showEvent );
  ldr.contentLoaderInfo.addEventListener( IOErrorEvent.IO_ERROR,
  showEvent );
  // 3. load()メソッドの実行
  ldr.load( url );
}
//プログレスバーの設定
function setProgressbar():void {
  this.bytes_progress.mode = ProgressBarMode.MANUAL;
  this.bytes_progress.minimum = 0;
  this.bytes_progress.setProgress( 0, 100 );
}
//イベントの発生を出力する関数
function showEvent(event:Event):void {
  switch ( event.type ) {
    case IOErrorEvent.IO_ERROR :                     ①
      percent_txt.text = "読込時にエラーが発生しました";
      break;
    case ProgressEvent.PROGRESS :
```

6
3

```
        this.bytes_progress.setProgress( ldr.contentLoaderInfo.
        bytesLoaded,  ldr.contentLoaderInfo.bytesTotal );
        percent_txt.text = int(bytes_progress.percentComplete) + "%
        読込済";
        break;
      case Event.COMPLETE :
        // 4. addChild()メソッドの実行
        this.addChild( ldr );
        //読み込んだコンテンツをステージの中央に配置
        ldr.x = ( stage.stageWidth - ldr.contentLoaderInfo.width )
        / 2;
        ldr.y = ( ( stage.stageHeight - 100 ) - ldr.contentLoader
        Info.height ) / 2
        //ボタンを非表示に(再読み込み防止)
        this.startLoading_btn.visible = false;
        percent_txt.text = "読込が完了しました";
        break;
    }
  }
  //プログレスバーの設定
  setProgressbar();                                              ①
```

このサンプルでは、進行状況とロードの完了をイベントで取得します。そのため、load()メソッドの実行後、addChild()メソッドを実行する部分は完了のイベントハンドラ内で行うようにしています。ここで使用するイベントはprogress(ロード進行中)とcomplete(ロードの完了)です。progressイベントハンドラ内で読み込み済みのサイズを取得して全体のロード状況とプログレスバーに表示します。これは①のイベントハンドラ内で行っています。このハンドラはイベントに共通で使用するため、event.typeプロパティにより、内部で処理を条件分けしています。

progressイベントのときはプログレスバーのsetProgress()メソッドにLoaderオブジェクトのcontentLoaderInforプロパティで参照できるLoaderInfoオブジェクトのbytesLoadedプロパティでロード済みバイト数を取得しています。

completeイベントはロードが完了しているので、addChild()メソッドでステージに配置し、ロードしたSWFのサイズを取得して位置を中央に配置しています。

7-7 モーショントゥイーン

7-7-1 Motionクラスを使う

ActionScriptでモーショントゥイーンを作成するときには、fl.motionパッケージのクラスを使用します。メインとなるのはMotionクラスとAnimatorFactoryクラスになります。MotionBaseクラスはMotionクラスの親クラスで、基本機能を提供します。トゥイーンの内容により、flash.geomパッケージやflash.filtersパッケージのクラスも使用します。ActionScriptによるモーショントゥイーンの作成手順は次のようになります。

ActionScriptによるモーショントゥイーンの作成手順

No.	手順	使用するプロパティ/メソッド
1	Motionオブジェクトを作成する	コンストラクタ
2	時間を設定する	durationプロパティ
3	アニメーションの基本となるプロパティを設定する	addPropertyArray()メソッド
4	AnimatorFactoryオブジェクトを作成する	コンストラクタにMotionオブジェクトを渡します。
5	アニメーションを適用するインスタンスを設定する	addTarget()メソッド

Motionクラスの主なプロパティ概要（MotionBaseクラスのプロパティを含む）

目的	プロパティ名	プロパティの概要
モーションの継続時間	duration :int	フレーム数でMotionインスタンスの継続時間を制御します。
3Dプロパティを持つモーションを作成	is3D :Boolean	モーションが3Dプロパティを含むかどうかを指定します。

Motionクラスの主なメソッド概要（MotionBaseクラスのメソッドを含む）

目的	メソッド名	メソッドの概要
モーションにフィルターを追加	initFilters	フィルターを初期化します。
	addFilterPropertyArray	キーフレームを基準としたフィルターの設定を行います。
キーフレームを追加	addKeyframe	Motionインスタンスにキーフレームを追加します。
モーションにプロパティの変更を追加	addPropertyArray	キーフレームを基準としたプロパティの設定を行います。このメソッドによるプロパティがアニメーションの基準になります。

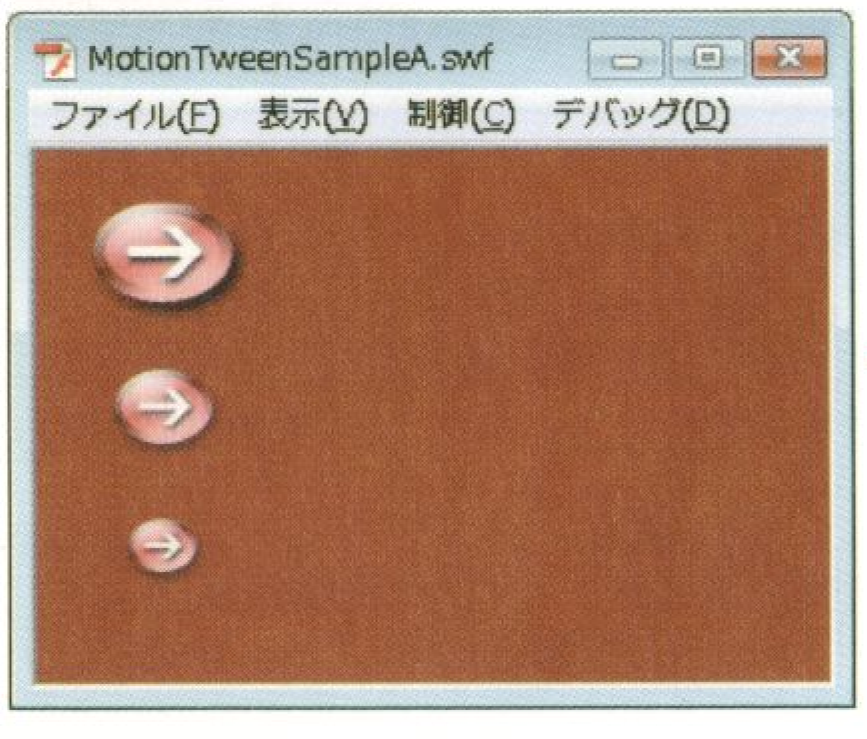
■ Motionクラスの主なメソッド概要（MotionBaseクラスのメソッドを含む）（続き）

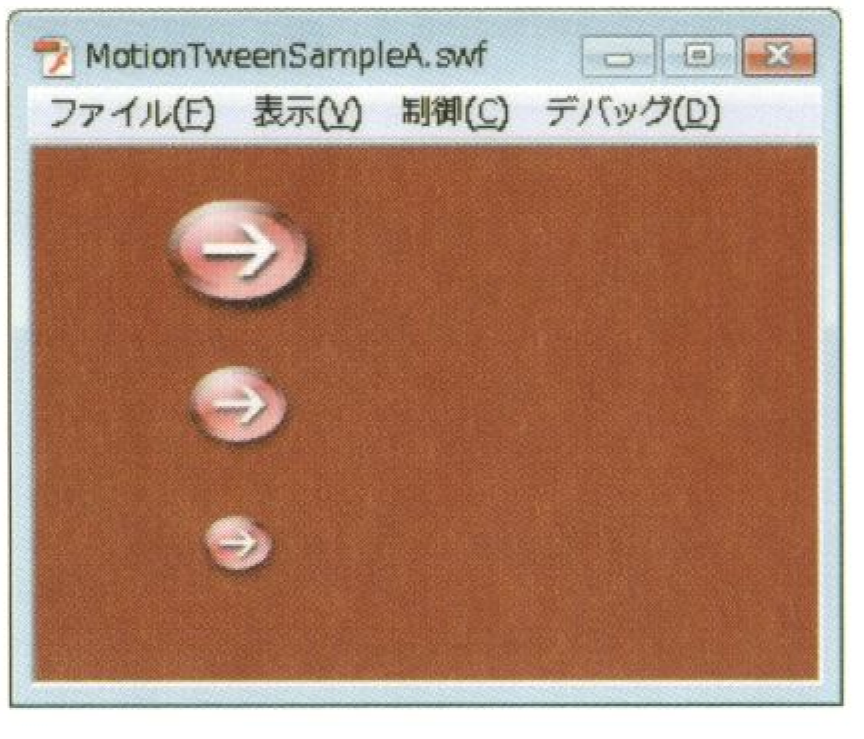
目的	メソッド名	メソッドの概要
アクティブなキーフレームを取得	getCurrentKeyframe	現在アクティブなキーフレームを取得できます。トゥイーン可能なプロパティ名を指定できます。
あるフレームのトゥイーンプロパティ値を取得	getValue	フレームインデックスを指定してトゥイーンプロパティ値を取得します。
あるフレームのトゥイーンプロパティ値を設定	getValue	フレームインデックスを指定してトゥイーンプロパティ値を設定します。

■ AnimatorFactoryクラスの主なメソッド概要（AnimatorFactoryBaseクラスのメソッドを含む）

目的	メソッド名	メソッドの概要
Motionオブジェクトを関連づける	AnimatorFactory	コンストラクタです。引数にMotionオブジェクトを指定して関連付けを行います。
Motionトゥイーンを表示オブジェクトに適用	addTarget	Motionトゥイーンを表示オブジェクトに適用します。

SAMPLE





SAMPLE CODE

Motionクラスによるシンプルなアニメーション MotionTweenSampleA.fla

```
import fl.motion.AnimatorFactory;
import fl.motion.MotionBase;
import fl.motion.Motion;

//1. Motionオブジェクトを作成
var myMotion:Motion = new Motion();
```

```
//2. 時間を設定
myMotion.duration = 5; ―――――――――――――――――――――――――――――――――――――――――――――― ①

//3. アニメーションの基本となるプロパティを設定
myMotion.addPropertyArray("x", [0, 10, 20, 30, 40] ); ――――――――― ②

//4. AnimatorFactoryオブジェクトを作成
var myAf:AnimatorFactory = new AnimatorFactory( myMotion );

//5. アニメーションを適用するインスタンスを設定
myAf.addTarget( target01 ); ┐
myAf.addTarget( target02 ); ├―――――――――――――――――――――――――――――――――――― ③
myAf.addTarget( target03 ); ┘
```


基本手順どおりのサンプルです。①で行っている時間の設定は、通常②で行うようなアニメーションプロパティ値の配列の要素数と等しい値にします。時間の設定については、7-2-2「モーショントゥイーンアニメーションを行う」を参照してください。②で行っているaddPropertyArray()メソッドでプロパティの変化を指示します。ここでは、xプロパティの値を0→10→20→30→40→0→10…と変化させる指定をしています。③でアニメーションを適用するインスタンスを設定していますが、ここではステージ上の3つのインスタンスに適用しています。target01〜target03はインスタンス名です。結果として、3つのアニメーションに同じプロパティの変化をもたらすアニメーションが適用されます。

7-7-2 モーショントゥイーンアニメーションを行う

MotionクラスとAnimatorFactoryクラスにより、トゥイーンアニメーションを作成できます。ここではプロパティの変化による、プロパティベースのトゥイーンを解説します。アニメーションの基本となるキーフレームのプロパティ変化はMotionクラスのaddPropertyArray()メソッドで行います。モーショントゥイーンの対象となる主なプロパティには、下表のものがあります。

モーショントゥイーンアニメーションの対象となるプロパティ

プロパティ	内容
x / y / z	親の座標空間での、オブジェクトの変形点の水平/垂直/深度（z軸）位置
scaleX/scaleY	変形点から適用されるオブジェクトの水平/垂直方向の拡大/縮小率（パーセンテージ）
skewX/skewY	変形点から適用されるオブジェクトの水平/垂直方向の傾斜角度
rotationX/ rotationY	x/y軸に対するオブジェクトの元の方向からの回転
rotationConcat	変形点から適用される、以前の方向に対する相対的な動作でのオブジェクトの回転（z 軸）の値
blendMode	オブジェクトの色と下にあるグラフィックの混合を指定するBlendModeクラスの値
matrix3D	キーフレームに指定されている場合は、そのmatrix3Dプロパティ。3Dトゥイーンに使用されます。使用されると、それまでのすべての変形プロパティは無視されます。
rotationZ	3D親コンテナを基準とする、元の方向からのオブジェクトのz軸の回転（度）。3DトゥイーンでrotationConcatの代わりに使用されます。

Motionクラスのアニメーションのためのプロパティ詳細

プロパティ	プロパティの詳細
duration :int	Motionオブジェクトの継続時間を表します。通常はこの値と等しい要素数のキーフレーム値の配列をaddPropertyArray()メソッドに渡してプロパティ値の変化を定義します。配列の要素数がdurationより多い場合、duration値はそれに合わせて調整されます。配列の要素数がdurationより小さい場合、配列の最後の要素の値が最後のフレームまで維持されます。

Motionクラスのアニメーションのためのメソッド詳細

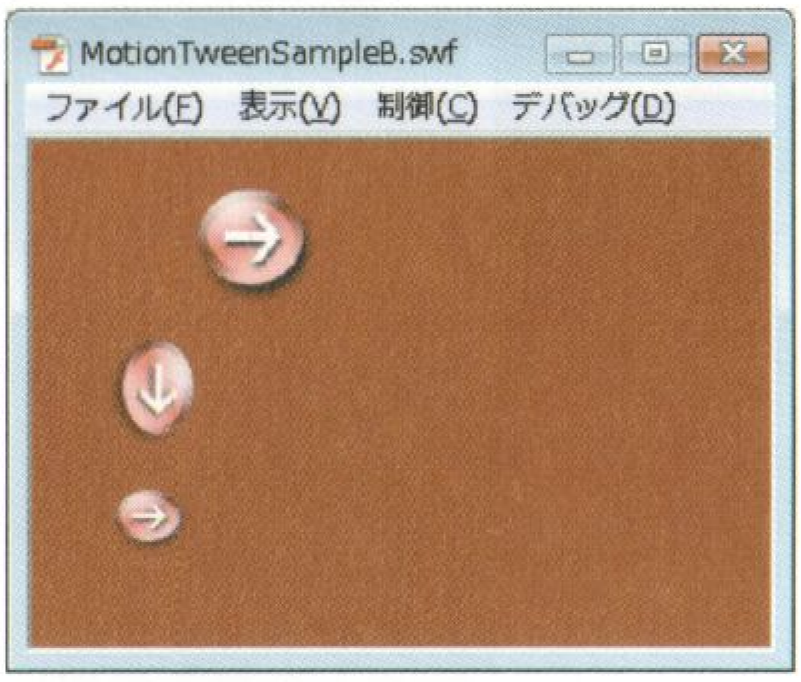
メソッド	メソッドの詳細
addPropertyArray(index:int, name:String, values:Array, startFrame:int = -1, endFrame:int = -1):void	Motionオブジェクトにアニメーションプロパティを設定します。 **name**：プロパティ名 **values**：nameで指定したプロパティの値の変化を表す配列です。配列内の値の順序によって、キーフレームへのそれぞれの値の割り当てが決まります。配列内のnull以外の値のそれぞれについて、キーフレームにプロパティ名／値のペアを格納します。 startFrame、endFrameで適用する開始/終了フレームを設定できます。

AnimatorFactoryクラスのメソッド詳細

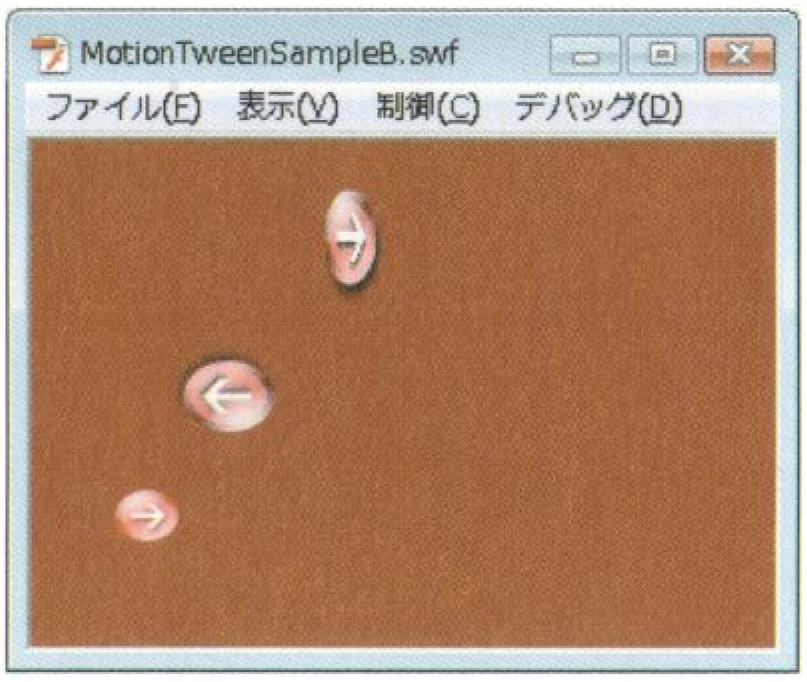
メソッド	メソッドの詳細
addTarget(target:DisplayObject, repeatCount:int = 0, autoPlay:Boolean = true, startFrame:int = -1, useCurrentFrame:Boolean = false):AnimatorBase	Motionオブジェクトを表示オブジェクトに適用します。このメソッドで指定された表示オブジェクトにMotionオブジェクトが持つアニメーショントゥイーンが実行されます。 **target**：トゥイーンを適用する表示オブジェクト **repeatCount**：アニメーションのリピート回数です。デフォルトの0では無限に繰り返します。 **autoPlay**：trueの場合、アニメーションを自動で開始します。 **startFrame**：親のタイムラインを基準とする、アニメーションが開始するフレームです。 **useCurrentFrame**：trueの場合、ターゲットオブジェクトを配置するアニメーションフレームを決定する親のcurrentFrameプロパティを使用します。

7
2

SAMPLE



→



SAMPLE CODE

複数のMotionオブジェクトを作成して表示オブジェクトに割り当てる MotionTweenSampleB.fla

```
import fl.motion.AnimatorFactory;
import fl.motion.MotionBase;
import fl.motion.Motion;
import flash.display.BlendMode;
```

```
//1. Motionオブジェクトを作成
var myMotion1:Motion = new Motion();
var myMotion2:Motion = new Motion();
var myMotion3:Motion = new Motion();
//2. 時間を設定
myMotion1.duration = 10;
myMotion2.duration = 10;
myMotion3.duration = 10;
//3. アニメーションの基本となるプロパティを設定                        ①
myMotion1.addPropertyArray("x", [0, 20, 40, 60, 80, 100, 120, 140,
160, 180] );
myMotion1.addPropertyArray("scaleX", [1, 1, 0.8, 0.8, 0.4, 0.4] );
myMotion2.addPropertyArray("x", [0,  0,   0,   0,  30,   60,  90,
120, 160, 180] );
myMotion2.addPropertyArray("rotationConcat", [0, 30, 90, 180, 180,
180, 90, 30, 0] );
myMotion3.addPropertyArray("x", [0,   0,  0,   0,   0,    0,    0,
60, 120, 180] );
myMotion3.addPropertyArray("blendMode", [BlendMode.NORMAL,
BlendMode.NORMAL,  BlendMode.NORMAL,  BlendMode.SCREEN,  BlendMode.
SCREEN,  BlendMode.SCREEN] );
//4. AnimatorFactoryオブジェクトを作成
var myAf1:AnimatorFactory = new AnimatorFactory( myMotion1 );
var myAf2:AnimatorFactory = new AnimatorFactory( myMotion2 );
var myAf3:AnimatorFactory = new AnimatorFactory( myMotion3 );
//5. アニメーションを適用するインスタンスを設定
myAf1.addTarget( target01, 1 );
myAf2.addTarget( target02, 2 );  ―――――――――――――――――――――②
myAf3.addTarget( target03, 3 );
```

基本手順にほんの少しアレンジを追加したサンプルです。Motionオブジェクトを複数作成しておき、プロパティをセットしておくことで、同じアニメーションを繰り返し使用することもできます。①でscaleX、rotationConcat、BlendModeなどのプロパティを使用しています。すべてのMotionオブジェクトでduration=10としていますが、addPropertyArray()の第2パラメータに渡す配列の要素数が10に満たない場合、最後の値が10フレーム目まで継続します。②でaddTarget()により、ターゲットオブジェクトを設定していますが、2番目のパラメータでアニメーションを繰り返す回数を指定しています。それぞれ1、2、3回実行して停止します。

7-7-3 モーショントゥイーンでフィルターアニメーションを作成する

Motionクラスと AnimatorFactoryクラスによるトゥイーンアニメーションにフィルターを追加する場合、アニメーションと同様に配列による指定を行います。指定は、事前にinitFilters()メソッドによる初期化を行い、addFilterPropertyArray()メソッドでフィルターの設定を行います。flash.filtersパッケージのフィルターを使用できます。

ActionScriptのフィルターによるモーショントゥイーンの作成手順

No.	手順	使用するプロパティ / メソッド
1	Motionオブジェクトを作成する	コンストラクタ
2	時間を設定する	durationプロパティ
3	使用するフィルターすべての完全修飾クラス名からなるストリング配列を作成し、初期化を行います。	initFilters()メソッド
4	アニメーションの基本となるフィルターを設定する	addFilterArray()メソッド
5	AnimatorFactoryオブジェクトを作成する	コンストラクタにMotionオブジェクトを渡します。
6	アニメーションを適用するインスタンスを設定する	addTarget()メソッド

flash.filtersパッケージのフィルタークラス

クラス	フィルターの内容
BevelFilter	表示オブジェクトにベベル効果を追加できます。
BitmapFilter	すべてのイメージフィルター効果の基本クラスです。
BitmapFilterQuality	BitmapFilterオブジェクトのレンダリング品質を設定する値が含まれます。
BitmapFilterType	BitmapFilterの型を設定する値が含まれます。
BlurFilter	表示オブジェクトにぼかし効果を適用できます。
ColorMatrixFilter	入力イメージの各ピクセルのRGBAカラー値とアルファ値に4×5マトリックス変換を適用することで、新しいRGBAカラー値とアルファ値からなる結果を作成できます。
ConvolutionFilter	マトリックス畳み込みフィルター効果を適用できます。
DisplacementMapFilter	指定されたBitmapDataオブジェクト（置き換えマップイメージといいます）のピクセル値を使用して、オブジェクトの置き換え（変位）を実行します。
DisplacementMapFilterMode	DisplacementMapFilterクラスのmodeプロパティの値を提供します。

flash.filtersパッケージのフィルタークラス（続き）

クラス	フィルターの内容
DropShadowFilter	ドロップシャドウを表示オブジェクトに追加します。
GlowFilter	表示オブジェクトにグロー効果を適用できます。
GradientBevelFilter	オブジェクトにグラデーションベベル効果を適用し、表示できます。
GradientGlowFilter	表示オブジェクトにグラデーショングロー効果を適用できます。
ShaderFilter	フィルター処理されるオブジェクトにシェーダを実行することによってフィルターを適用します。

Motionクラスのフィルターのためのプロパティ詳細

プロパティ	プロパティの詳細
duration :int	Motionオブジェクトが持つフレーム数を表します。通常はこの値を最大値とするフィルタープロパティ値の配列をaddFilterPropertyArray()メソッドの第3パラメータに渡してフィルタープロパティ値の変化を定義します。配列の要素数がdurationより小さい場合、配列の最後の要素の値が最後のフレームまで維持されます。

Motionクラスのフィルターアニメーションのためのメソッド詳細

メソッド	メソッドの詳細
initFilters (filterClasses:Array, gradientSubarrayLengths: Array, startFrame:int = -1, endFrame:int = -1) :void	Motionオブジェクトに適用するフィルタークラスを定義します。適用するフィルタークラス名を配列にしてパラメータに渡します。 **filterClasses**：使用するフィルターの完全修飾クラス名をStringで配列にして渡します。 **gradientSubarrayLengths**：使用するフィルターがGradientGlowFilterまたはGradientBevelFilterである場合、コンストラクタのcolors,alphas,rationsパラメータを初期化する配列の長さを指定します。 **startFrame/endFrame**：初期化を適用するフレームを指定する場合、Motionオブジェクトの開始/終了フレームを対象にします。デフォルトは全フレーム対象です。
addFilterPropertyArray (index:int, name:String, values:Array, startFrame:int = -1, endFrame:int = -1):void	Motionオブジェクトにフィルタープロパティを設定します。 **index**：initFilters()メソッドの第1パラメータに設定した配列のインデックスにより、使用するフィルターを指定します。例えばinitFilters(["flash.filters.BevelFilter"、"flash.filters.GlowFilter"],..)とした場合、0：BevelFilter、1：GlowFilterを表します。 **name**：使用するフィルターのプロパティ名を指定します。 **values**：Motionオブジェクトのキーフレームごとに適用するフィルタープロパティ値を配列にします。配列の要素数がdurationプロパティで指定された範囲より小さい場合、最後の要素の値が継続されます。 startFrame、endFrameで適用する開始/終了フレームを設定できます。開始フレームは0、-1を指定すると先頭フレームから適用されます。

AnimatorFactoryクラスのメソッド詳細

メソッド	メソッドの詳細
`addTarget (target:DisplayObject, repeatCount:int = 0, autoPlay:Boolean = true, startFrame:int = -1, useCurrentFrame:Boolean = false ):AnimatorBase`	Motionオブジェクトを表示オブジェクトに適用します。このメソッドで指定された表示オブジェクトにMotionオブジェクトが持つフィルターが実行されます。 **target**：トゥイーンを適用する表示オブジェクト **repeatCount**：フィルターのリピート回数です。0（デフォルト）では無限に繰り返します。 **autoPlay**：trueの場合、フィルタートゥイーンを自動で開始します。 **startFrame**：親のタイムラインを基準とする、アニメーションが開始するフレームです。 **useCurrentFrame**：trueの場合、ターゲットオブジェクトを配置するアニメーションフレームを決定する親のcurrentFrameプロパティを使用します。

SAMPLE



SAMPLE CODE

フィルターによるアニメーション

MotionTweenSampleC.fla

```
import fl.motion.AnimatorFactory;
import fl.motion.MotionBase;
import fl.motion.Motion;
import flash.filters.*;

//1. Motionオブジェクトを作成
var myMotion1:Motion = new Motion();

//2. 時間を設定
myMotion1.duration = 8;

//3. フィルターを初期化
myMotion1.initFilters(["flash.filters.DropShadowFilter", "flash.
```

```
filters.BlurFilter"], [0, 0, 0]);                ①

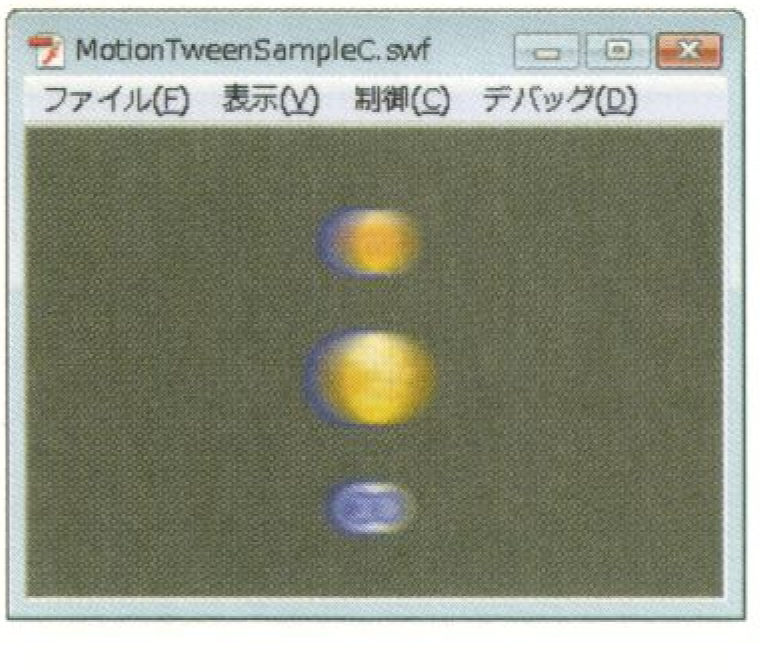
//4. アニメーションの基本となるプロパティを設定
//myMotion1.addFilterPropertyArray(0, "angle", [0, 20, 40, 60,
80, 160, 220, 280, 330, 340, 350, 360]);
myMotion1.addFilterPropertyArray(0, "angle", [0, 180, 0, 180, 0,
270, 0, 90]);
myMotion1.addFilterPropertyArray(0, "color", [0xFF0000, 0x00FF00,   ②
0x444444, 0x0000FF, 0x888888, 0x660000, 0x444444, 0x006600]);
myMotion1.addFilterPropertyArray(1, "blurX", [0, 2, 8, 16, 0]);
myMotion1.addFilterPropertyArray(1, "blurY", [0, 0, 0, 0, 0, 2,
8, 16, 0]);

//5. AnimatorFactoryオブジェクトを作成
var myAf1:AnimatorFactory = new AnimatorFactory( myMotion1 );

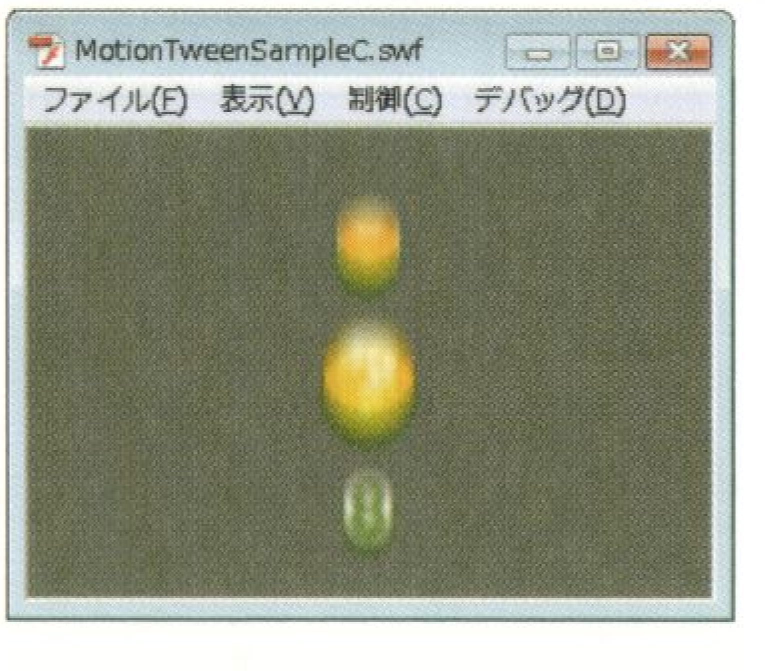
//6. アニメーションを適用するインスタンスを設定
myAf1.addTarget( target01 );
myAf1.addTarget( target02 );                     ③
myAf1.addTarget( target03 );
```

①のinitFilters()メソッドで使用する2つのフィルター（DropShadowFilter、BlurFilter）を初期化しています。②でフィルタープロパティの設定値によるアニメーションを作成しています。addFilterPropertyArray()メソッドの第1パラメータが0の場合、DropShadowFilterが1の場合、BlurFilterのプロパティ設定になります。③でステージ上の3つのオブジェクトにアニメーションを適用しています。

■ BlurFilter.blurXプロパティによる変化



■ BlurFilter.blurYプロパティによる変化



7-8 IK（インバース・キネマティックス）

7-8-1 fl.ikパッケージを使う

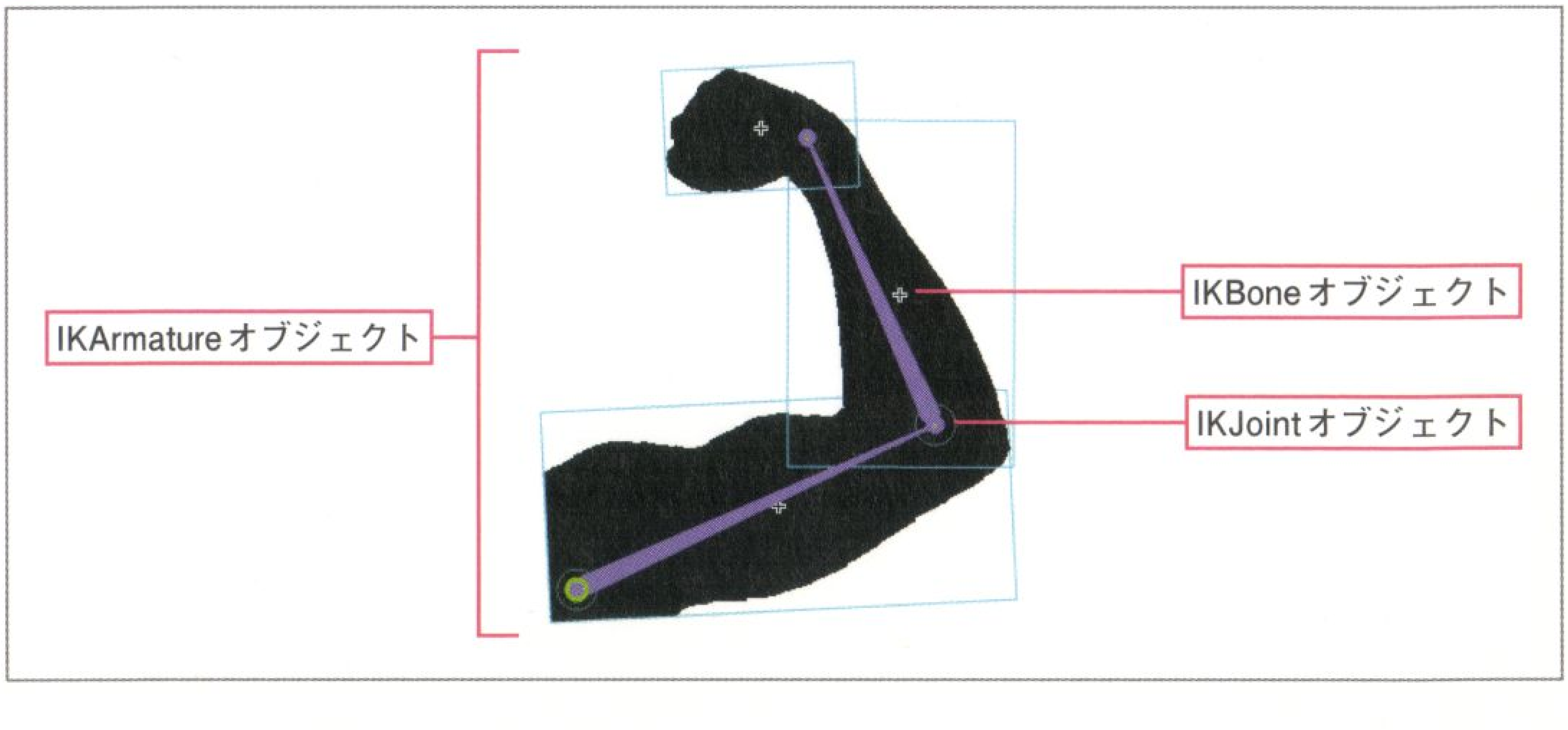
Flash CS4から利用可能になったIK（インバース・キネマティックス）は骨（ボーン）と関節で構成される自然なアニメーションを提供するテクノロジーです。ActionScriptでIKの制御を行う場合、まずオーサリング環境でアーマチュア（つながったボーン全体）を作成します。ActionScriptによるIKオブジェクトの作成はできません。ActionScriptでIKの制御を行うためにfl.ikパッケージが用意されています。IKArmatureクラスはアーマチュアオブジェクトを表すクラスで、IKArmatureオブジェクトはボーンとジョイントで構成されますが、ボーンはIKBone、ジョイントはIKJointクラスのインスタンスになります。IKManagerクラスはアーマチュアオブジェクトの情報を取得するためのプロパティとメソッドを提供します。ActionScriptでアーマチュアの情報を参照するには「実行時」オプションを設定しておく必要があります。

8
1

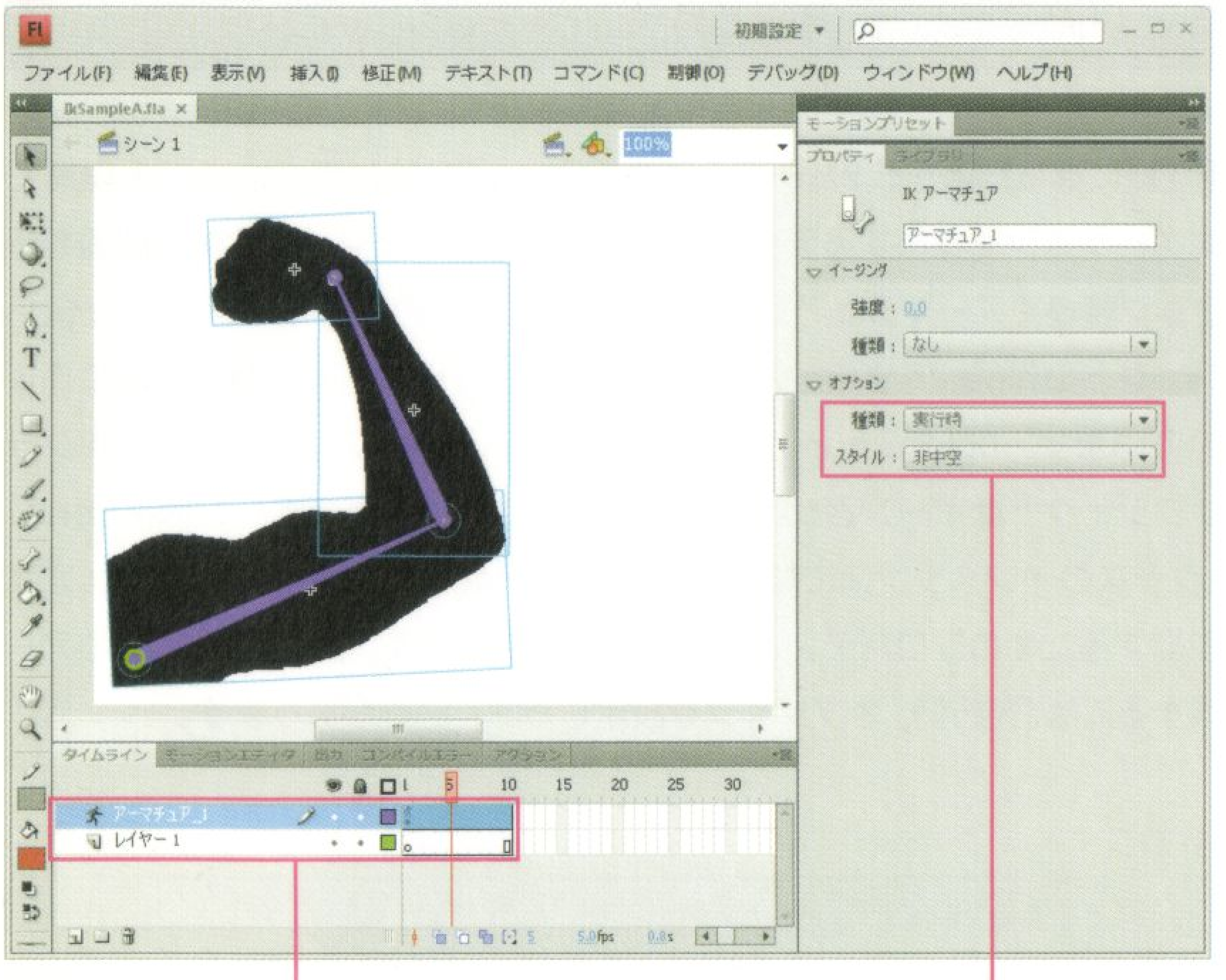
fl.ikパッケージのクラス

クラス	内容
IKManager	すべてのアーマチュアオブジェクトを管理します。
IKArmature	インバース・キネマティックス（IK）アーマチュアを表します。アーマチュアはFlashオーサリングツールでのみ作成することができます。
IKBone	アーマチュアの基本コンポーネントセグメントである「ボーン」を表します。ボーンはIKJointオブジェクトで互いに接続されます。
IKEvent	アーマチュアに関するイベントを定義します。
IKJoint	2つのボーンの接続部を定義し、ボーンの接続方法や動作に関するプロパティを提供します。
IKMover	アーマチュアの動作に関するプロパティと実行するメソッドを提供します。

IKの主なオブジェクト



■ オーサリング時のアーマチュアオブジェクトのオプション設定



①ボーンツールで作成されたアーマチュアレイヤーを選択

②[オプション]タブの[種類]を「実行時」にする

IKManagerクラスの主なプロパティ概要

目的	プロパティ名	プロパティの概要
存在するアーマチュア数を調べる	numArmatures :int	ステージ上のアーマチュアの数を表します。

IKManagerクラスの主なメソッド概要

目的	メソッド名	メソッドの概要
インデックスを指定してアーマチュアオブジェクトを取得する	getArmatureAt(index:int):IKArmature	[static]指定のインデックス位置のアーマチュアを返します。
名前を指定してアーマチュアオブジェクトを取得する	getArmatureByName(name:String):IKArmature	[static]指定された名前を持つアーマチュアを返します。

IKManagerクラスの主なメソッド概要(続き)

目的	メソッド名	メソッドの概要
アーマチュアのライブトラッキングを有効/無効にする	trackAllArmatures(enable:Boolean=true):void	すべてのアーマチュアのライブトラッキングを有効または無効にします。
	trackIKArmature(theTree:IKArmature, enable:Boolean = true):void	指定されたIKArmatureインスタンスのライブトラッキングを有効または無効にします。
	trackIKObject(target:DisplayObject, enable:Boolean = true):*	指定されたIK表示オブジェクトインスタンスのライブトラッキングを有効または無効にします。

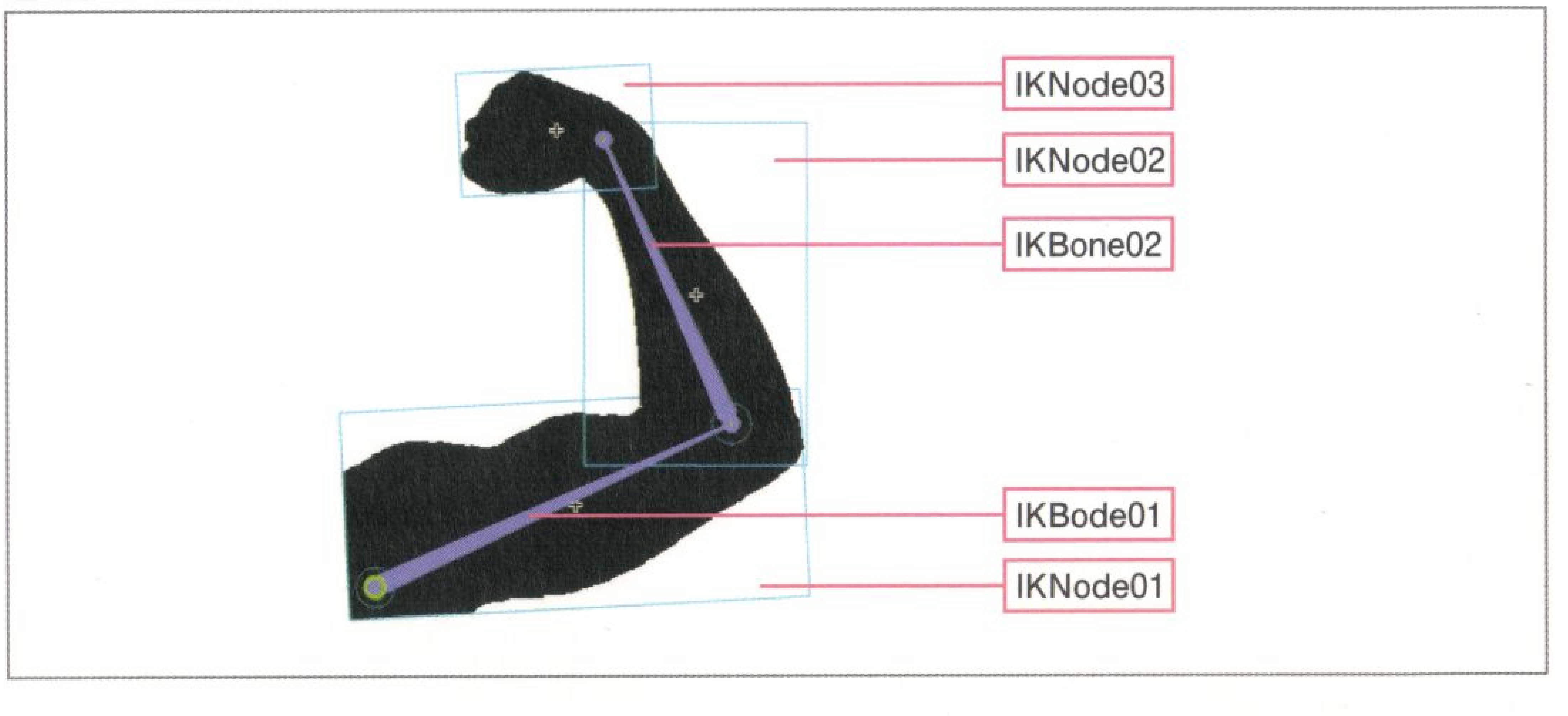
IKArmatureクラスの主なプロパティ概要

目的	プロパティ名	プロパティの概要
アーマチュアの名前	name:String	このアーマチュアオブジェクトの名前を表します。
ルートジョイントを参照	rootJoint:IKJoint	このアーマチュアオブジェクトのルートジョイント(関節)を表します。

IKArmatureクラスの主なメソッド概要

目的	メソッド名	メソッドの概要
ボーンを名前で取得する	getBoneByName(targetName:String):IKBone	targetNameで指定したボーンオブジェクトを取得します。
ライブラリシンボルインスタンスのIK移動をアクティブ化	registerElements(stage:DisplayObjectContainer):*	アーマチュアを含むMovieClipライブラリを実行時にインスタンス化する場合、このメソッドを呼び出します。

SAMPLE



8
1

SAMPLE CODE

ステージ上のアーマチュアの数と名前リストを取得する IKSampleB.fla

```
import fl.ik.*;
var arm:IKArmature;
var rootJoint:IKJoint;
var currentJoint:IKJoint;
var bone:IKBone;
var numChildren_:int;
//IKArmatureオブジェクトを調べる関数
function getArmature():void {
  var i:int;
  trace( "IKArmatureオブジェクトの数:" + IKManager.numArmatures );  ――①
  for ( i = 0 ; i  < IKManager.numArmatures ; i++ ) {   ┐
    arm = IKManager.getArmatureAt( i );                 │
    //ルートジョイントオブジェクトを取得                 │
    rootJoint = arm.rootJoint;                          ├―②
    getJoints( rootJoint );                             │
  }                                                     ┘
}
//IKJointオブジェクトとIKBoneオブジェクトを調べる関数
function getJoints( j:IKJoint ):void {  ――③
  var i:int;
  trace( "¥nジョイント名:" + j.name + ", numChildren=" + j.numChildren );
  bone = j.bone;                    ┐
  trace( "ボーン名:" + bone.name );  ┘―④
  for ( i = 0 ; i < j.numChildren ; i++ ) {
    currentJoint =  IKJoint( j.getChildAt( i ) );
    if ( currentJoint.numChildren == 0 ) {                                     ┐
      trace( "¥nジョイント名:" + currentJoint.name + ", numChildren="          │
      + currentJoint.numChildren );                                            ├―⑤
      trace( "ボーン名:" + currentJoint.bone.name );                            │
    }                                                                          ┘
    if ( IKJoint( j.getChildAt( i ) ).numChildren > 0 ) {
      getJoints( IKJoint( j.getChildAt( i ) ) );
    }
  }
}
getArmature();
stop();
```

ステージ上のIKArmatureオブジェクトとその関連するオブジェクトを抽出するサンプルです。まず①で行っているように、IKAamatureオブジェクトの数を調べるには、numChildrenプロパティを使用します。②のfor文ですべてのオブジェクトについて調べています。各IKArmatureオブジェクトにはルートジョイントオブジェクトがあり、それを元にして関連するジョイントとボーンを調べることができます。IKArmatureオブジェクトのrootJointプロパティでルートとなるIKJointオブジェクトを取得できます。ルートジョイントから関連する他のジョイント、ボーンを抽出するには、getChidAt()メソッドで調べることができます。そのための関数は③以降のgetJoints()関数です。ジョイントに関連づけされたボーンは、④で行っているように、IKJointオブジェクトのboneプロパティで調べることができます。getJoints()関数内では、パラメータで与えられたジョイントが子を持つ場合、再び同じgetJoints()関数を呼び出して処理していますが、最後のジョイントではnumChildrenプロパティが0となるため、⑤のように情報を出力するようにしています。以上の処理により、IKArmatureオブジェクト内のIKJoint、IKBoneオブジェクトを抽出しています。このサンプルでは、ステージ上に作成したオブジェクトに次のような名前をつけています。IKNodeXXという名前をつけたノード名と同じ名前がジョイント名として参照されます。

▶ 実行結果（トレース出力）

```
IKArmatureオブジェクトの数:1

ジョイント名:ikNode01, numChildren=1
ボーン名:ikBoneName9

ジョイント名:ikNode02, numChildren=1
ボーン名:ikBone01

ジョイント名:ikNode03, numChildren=0
ボーン名:ikBone02
```

[オプション] タブの [種類] を「オーサリング時」で実行した場合、次のような結果になります。

```
IKArmatureオブジェクトの数:0
```

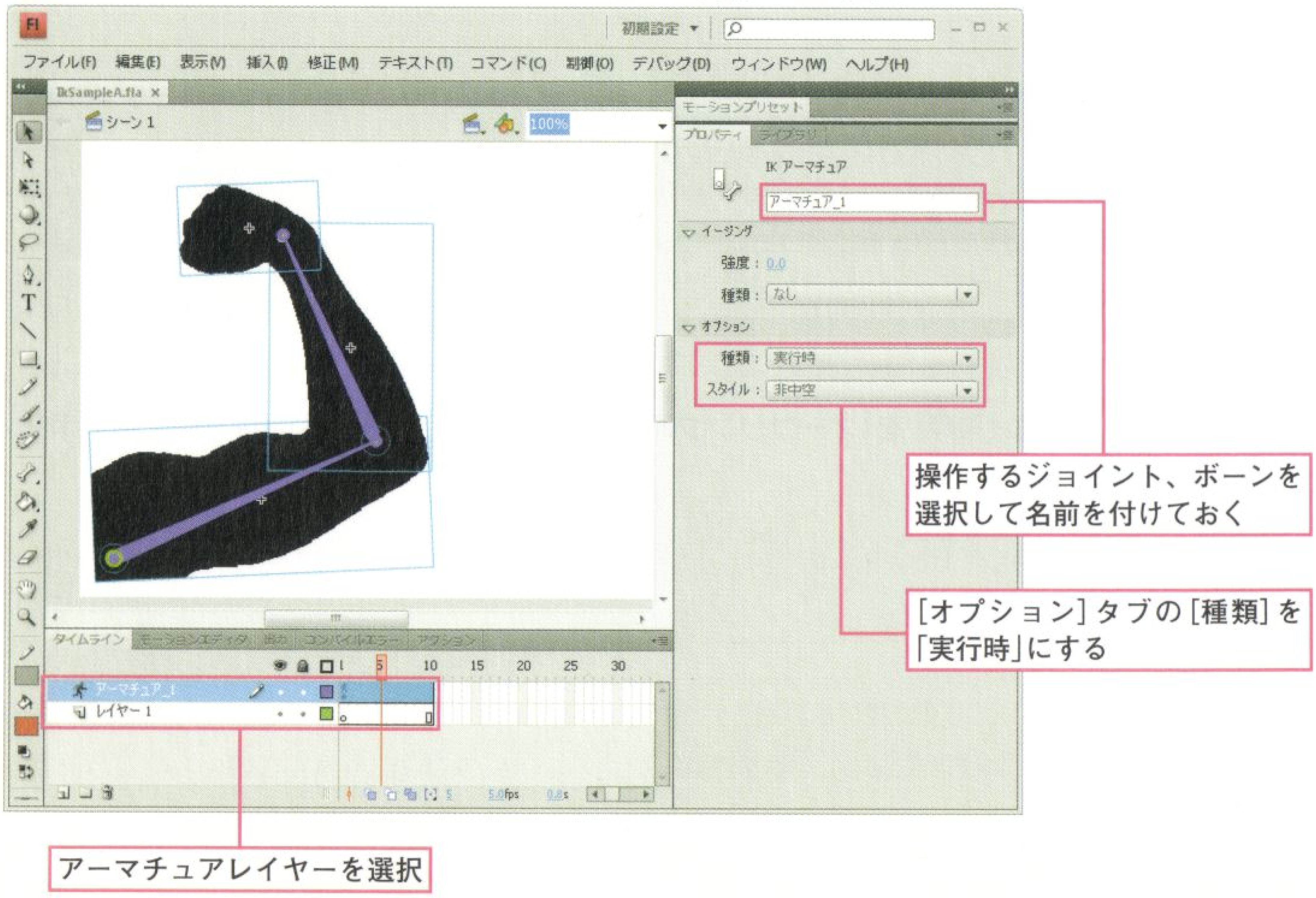
7-8-2 ActionScriptでアーマチュアを動かす

fl.ikパッケージのクラスを利用してアーマチュアオブジェクトを取得し、動かすことができます。7-8-1で解説したように、作成時に名前をつけておくとその名前でオブジェクトを参照することができます。名前はオーサリング中にステージ上のオブジェクトを選択してプロパティインスペクタで入力します。また、ActionScriptでアーマチュアを動かすためには「実行時」オプションを設定しておく必要があります。手順は次のようになります。

ActionScriptでIKの制御を行う手順

No.	作業内容	使用する主なクラス
1	オーサリング環境でボーンツールによりアーマチュアを作成して名前をつける。プロパティインスペクタで「実行時」オプション設定を行う（下図参照）。	—
2	アーマチュアの名前を参照してIKArmatureオブジェクトを作成する	IKArmature IKManager
3	アーマチュアオブジェクトから骨と関節に対応する変数を宣言する	IKBone IKJoint
4	動きを制御するためのIKMoverオブジェクトを生成する	IKMover IKEvent
5	IKMoverオブジェクトのプロパティとメソッドでアーマチュアを動かす	

オーサリング時のアーマチュアオブジェクトの名前づけと設定



IKBoneクラスの主なプロパティ概要

目的	プロパティ名	プロパティの概要
ボーンの名前	name :String	このボーンオブジェクトの名前を表します。
ヘッドジョイント/テールジョイントを参照	headJoint :IKJoint tailJoint :IKJoint	このボーンの2つのジョイントのうち、ルートに近いヘッドジョイントと逆側のテイルジョイントを表します。

IKJointクラスの主なプロパティ概要

目的	プロパティ名	プロパティの概要
ジョイントが関連するボーン	bone :IKBone	このジョイントオブジェクトが関連するボーンを表します。
ジョイントオブジェクトの名前	name :String	このジョイントオブジェクトの名前を表します。
ジョイントの位置する現在位置座標	position :Poit	このジョイントオブジェクトの座標を表します。座標を直接変更することはできません。
回転の現在値	rotation :Number	現在の回転値を表します。回転値はオブジェクトの移動により、自動的に決まります。
回転を制限されているか	rotationContrained :Boolean	回転の最大値/最長値が設定されているかを示します。
回転が有効か	rotationEnabled :Boolean	回転が有効か無効かを示します。
回転の最大値/最小値	rotationMax/rotationMin :Number	回転の最大値/最長値を表します。
回転速度を制限	speed :Number	ボーンの動きの速度をパーセント値で制限します。デフォルトは100 (%) で、これは無制限を表します。
x方向/y方向への現在の移動値	xTranslation/yTranslation :Number	x方向/y方向への移動の現在値を表します。直接このプロパティの値を変更できません。
x方向/y方向への移動が制限されているか	xTranslationconstrained/ yTranslationConstrained: Boolean	x方向/y方向への移動最大値/最小値が設定されているかを示します。
x方向/y方向への移動が有効か	xTranslationEnabled/ yTranslationEnabled :Boolean	x方向/y方向への移動が有効/無効を示します。
x方向/y方向への移動最大値/最小値	xTranslationMax/ yTranslationMax :Number	x方向/y方向への移動最大値/最小値を示します。
ジョイント動作の制限の種類を表す定数	DDF_ROTATION:int = 1 DDF_XTRANS:int = 2 DDF_YTRANS:int = 4	回転を制限することを示します。 X方向への移動を制限することを示します。 Y方向への移動を制限することを示します。

IKJointクラスの主なメソッド概要

目的	メソッド名	メソッドの概要
ジョイントの動きを制限	setConstrained(dof:int, enable:Boolean):*	dofで指定した要素に対する制限を有効/無効にします。dofはIKJointクラスの定数で指定できます。
ジョイントの動きを有効/無効	setEnabled(dof:int, enable:Boolean):*	dofで指定した要素に対する有効/無効を設定します。dofはIKJointクラスの定数で指定できます。
ジョイントの動きが制限されているか	isEnabled(dof:int):*	dofで指定した自由度の設定が有効になっているかどうかを返します。

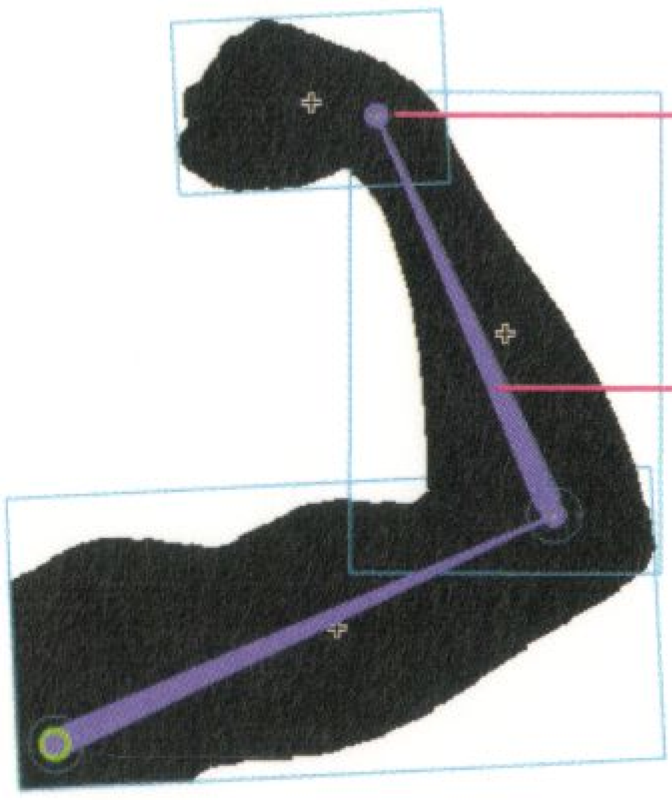
IKMoverクラスの主なプロパティ概要

目的	プロパティ名	プロパティの概要
IKジョイントを移動できる最大距離を制限	distanceLimit : Number	IKエンジンがIKジョイントを各反復処理で移動する最大距離(ピクセル単位)です。
1回の移動での最大繰り返し回数を制限	iterationLimit : int	IKエンジンのIKジョイントの1回の移動での最大繰り返し回数を指定します。
移動を実行する最大時間(ミリ秒単位)を指定	timeLimit : int	IKエンジンがIKジョイント移動を実行する最大時間(ミリ秒単位)を指定します。
distanceLimit値を超えた後、IKエンジンが移動を終了するかどうか	limitByDistance : Boolean	distanceLimit値を超えた後、IKエンジンが移動を終了するかどうかを指定します。
iterationLimit値を超えた後、IKエンジンが移動を終了するかどうか	limitByIteration : Boolean	iterationLimit値を超えた後、IKエンジンが移動を終了するかどうかを指定します。
指定されたtimeLimitを超えた後、IKエンジンが移動を終了するかどうか	limitByTime : Boolean	指定されたtimeLimitを超えた後、IKエンジンが移動を終了するかどうかを指定します。

IKMoverクラスの主なメソッド概要

目的	メソッド名	メソッドの概要
IKMoverオブジェクトをIKJointオブジェクトと関連づける	IKMover(endEffector:IKJoint, targetIn:Point)	コンストラクタの第1パラメータにIKJointオブジェクトを渡して関連づけできます。
IKジョイントを移動する	moveTo(targetIn:Point):*	すべての関連するIKJointオブジェクトをIK制約に基づいて更新し、関連付けられたIKJointオブジェクトをターゲットポイントに移動させます。

SAMPLE



SAMPLE CODE

IK アーマチュアを動かす IKSampleB.fla

```
import fl.ik.*;

var arm:IKArmature;
var targetBone:IKBone;
var targetJoint:IKJoint;
var ikm:IKMover;

function init():void {
  arm = IKManager.getArmatureAt( 0 );
  //名前でIKBoneオブジェクトを取得
  targetBone = arm.getBoneByName( "targetBone" );                    ①
  //IKBoneオブジェクトのルートと逆側のIKJointオブジェクトを取得
  targetJoint = targetBone.tailJoint;
  //targetJointの現在の位置を基点とするIKMoverオブジェクトを作成
  ikm = new IKMover( targetJoint, targetJoint.position );           ②
  //「伸ばす」ボタンと「曲げる」ボタンのイベントリスナー登録
  b1_btn.addEventListener( MouseEvent.CLICK, moveForward );
  b2_btn.addEventListener( MouseEvent.CLICK, moveBack );
}

//「伸ばす」アクション
function moveForward(event:MouseEvent):void {                        ③
  var pt:Point = targetJoint.position;
```

```
  //ジョイントの位置をx:10、y:10移動させる
  ikm.moveTo( new Point( pt.x + 10, pt.y + 10 ) );      ③
}

//「曲げる」アクション
function moveBack(event:MouseEvent):void {
  var pt:Point = targetJoint.position;
  //ジョイントの位置をx:-10、y:-10移動させる               ④
  ikm.moveTo( new Point( pt.x - 10, pt.y - 10 ) );
}
init();
stop();
```


サンプルでは、3つのジョイントのうち、先端のこぶし部分の位置をプログラムで変更します。アーマチャアオブジェクトの先端部分を動かすとルート方向のボーンとジョイントが連動して動作します。こぶし部分のジョイントオブジェクトを得るために必要となるのは、そのジョイントに連結されたボーンの名前です。これは、IKArmatureクラスにgetBoneByName()メソッドがあり、BoneクラスにtailJointプロパティがあるためです。

①でそれらによるジョイントオブジェクトの取得を行っています。ターゲットのジョイントが取得できたら、IKMoverオブジェクトを生成します。

②で行っているように、コンストラクタの第1パラメータにターゲットのジョイント、第2パラメータに基準となる最初の位置をPointオブジェクトで指定します。ここでは、現在のジョイントの位置を、IKJointオブジェクトのPositionプロパティで取得して設定しています。実際のアクションは、③と④の関数で行っています。IKMoverオブジェクトのmoveTo()メソッドで、動かす先の座標を指定すると、目的のジョイントが移動し、関連するボーンとジョイントが連動します。


IkSampleB.swf
ファイル(F) 表示(V) 制御(C) デバッグ(D)
伸ばす
曲げる


IkSampleB.swf
ファイル(F) 表示(V) 制御(C) デバッグ(D)
伸ばす
曲げる


IkSampleB.swf
ファイル(F) 表示(V) 制御(C) デバッグ(D)
伸ばす
曲げる

7-8-3 IKEventを使う

PART 1 2 3 4 5 6 7 8 9

IKMoverオブジェクトによるIKオブジェクトの移動に伴い、IKイベントが発生します。イベントはIKMoverオブジェクトのプロパティで指定した移動距離、回数、時間を超過して移動を終了したときに発生します。

IKEventクラスの主なイベント(定数)とイベント内容

イベント(定数)	イベントの内容(関連するIKMoverのプロパティ)
DISTANCE_LIMIT	IKMover.distanceLimitで指定された距離を超えた場合に発生します
ITERATION_LIMIT	IKMover.iterationLimitで指定された繰り返し回数を超えた場合に発生します
SINGLE_STEP	IKMoverの反復処理の各ステップで発生します。
TIME_LIMIT	IKMover.timeLimitで指定された距離を超えた場合に発生します

IKEventクラスの主なプロパティ概要

目的	プロパティ名	プロパティの概要
ジョイントの元の位置からの距離	distance :Number	ジョイントの元の位置からの距離(ピクセル単位)を表します。
IK移動実行の繰り返し回数	iterationcount :int	IK移動実行の繰り返し回数を表します。
イベントに関連したジョイント	joint :IKJoint	イベントに関連したジョイントを表します。
前回のIKイベントからの経過時間	time :int	前回のIKイベント送出からの経過時間(ミリ秒単位)を表します。

SAMPLE CODE

IKイベントを使用する

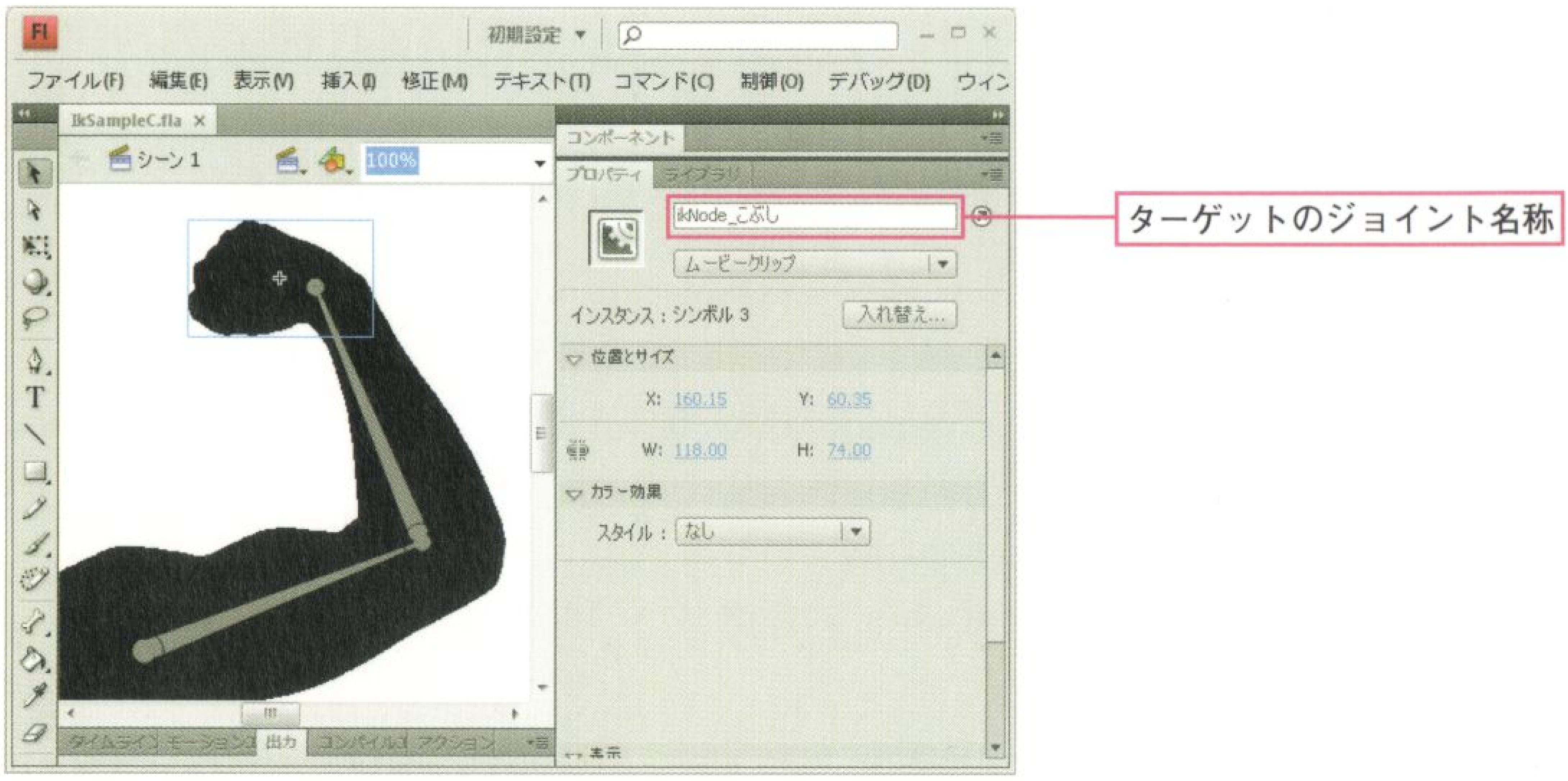
IKSampleC.fla

```
import fl.ik.*;
var arm:IKArmature;
var targetBone:IKBone;
var targetJoint:IKJoint;
var ikm:IKMover;
function init():void {
  arm = IKManager.getArmatureAt( 0 );
  //名前でIKBoneオブジェクトを取得
  targetBone = arm.getBoneByName( "targetBone" );
```

```
  //IKBoneオブジェクトのルートと逆側のIKJointオブジェクトを取得
  targetJoint = targetBone.tailJoint;
  //targetJointの現在の位置を基点とするIKMoverオブジェクトを作成
  ikm = new IKMover( targetJoint, targetJoint.position );
  //IKMoverオブジェクトに距離の制限プロパティを設定 ―①
  ikm.distanceLimit = 0.25; ―②
  //IKMoverオブジェクトに距離の制限値を超えた場合停止するプロパティを設定
  ikm.limitByDistance = true; ―③
  //IKMoverオブジェクトが距離の制限値を超えた場合のイベントハンドラを設定
  ikm.addEventListener(IKEvent.DISTANCE_LIMIT, limitExceeded ); ―④
  //「伸ばす」ボタンと「曲げる」ボタンのイベントリスナー登録
  b1_btn.addEventListener( MouseEvent.CLICK, moveForward );
  b2_btn.addEventListener( MouseEvent.CLICK, moveBack );
}
//「伸ばす」アクション
function moveForward(event:MouseEvent):void {
  var pt:Point = targetJoint.position;
  //ジョイントの位置をx:10、y:10移動させる
  ikm.moveTo( new Point( pt.x + 10, pt.y + 10 ) );
}
//「曲げる」アクション
function moveBack(event:MouseEvent):void {
  var pt:Point = targetJoint.position;
  //ジョイントの位置をx:-10、y:-10移動させる
  ikm.moveTo( new Point( pt.x - 10, pt.y - 10 ) );
}
//IKMoverのイベントハンドラ
function limitExceeded(event:IKEvent):void {
  trace( "IKEventが発生しました。" + event.type );
  trace( "対象のジョイント:" + event.joint.name );
  trace("distance : " + event.distance ); ―⑤
  trace("iterationCount:" + event.iterationCount );
  trace("¥n" );
}
init();
stop();
```

8
3

7-8-2 のサンプルコードにIKMoverに対する距離の制限を行い、イベントを発生させています。①からがその設定です。その中の②でdistanceLimitプロパティを設定しており、ここでIKMoverオブジェクトに距離の制限をかけます。距離の制限を超えたときに移動を停止するためには、③のようにlimitByDistanceプロパティをtrueにする必要があります。距離の制限値を超えた場合、distanceLimitイベントが発生するので、④でハンドラの登録を行っています。このとき、イベントオブジェクトはIKEventになります。⑤のハンドラ関数でそのプロパティを参照しています。



▶ 実行結果（トレース出力：「伸ばす」ボタンを３回押した場合）

```
IKEventが発生しました。distanceLimit
対象のジョイント:ikNode_こぶし
distance : 0.19317374356947167
iterationCount:6

IKEventが発生しました。distanceLimit
対象のジョイント:ikNode_こぶし
distance : 0.1694199953439017
iterationCount:6

IKEventが発生しました。distanceLimit
対象のジョイント:ikNode_こぶし
distance : 0.14397025475594957
iterationCount:6
```

7-9-1 2次元の平面座標でのオブジェクト操作をサポートするクラス

2次元の平面座標系で表示オブジェクトを移動、回転したり、距離を測定するなどの操作をサポートするクラスはジオメトリ(geometry:幾何学)クラスとして、flash.geomパッケージに用意されています。そのクラス群は直接オブジェクトへの操作を実行するのではなく、座標自体を表したり、距離を算出したり、移動に伴う座標の変換(変更)を行うというような補助的な役割を提供します。2次元平面での座標位置は、ステージの左上を基準(x,y) = (0,0)として、xの値が増加すると右方向、yの値が増加すると下方向に移動します。ActionScript3.0に用意されているジオメトリ関連のクラスには下表のものがあります。もっとも基本となるPointクラスは平面上の座標を表し、座標に関する演算メソッドを提供します。

2次元関連機能をサポートするActionScript3.0クラス

パッケージ	クラス	内容
flash.geom	Point	x、yからなる2次元座標の位置を表します。
	Rectangle	左上の座標と幅、高さを持つ四角形の領域を表します。
	Matrix	2次元平面で回転、拡大/縮小、平行移動などの操作を行うための座標変換を行うメソッドを提供します。
	Tramsform	表示オブジェクトがMatrixインスタンスにアクセスする方法を提供します。3DMatrixオブジェクトやcolorTransformへのアクセスも提供します。

Pointクラスの主なプロパティ概要

目的	プロパティ名	プロパティの概要
オブジェクトの水平/垂直座標取得/設定する	x:Number y:Number	Pointオブジェクトが持つ水平/垂直座標を表します。
(0, 0)からの直線距離を求める	length :Number	現在の座標の(0, 0)からの距離を算出します。

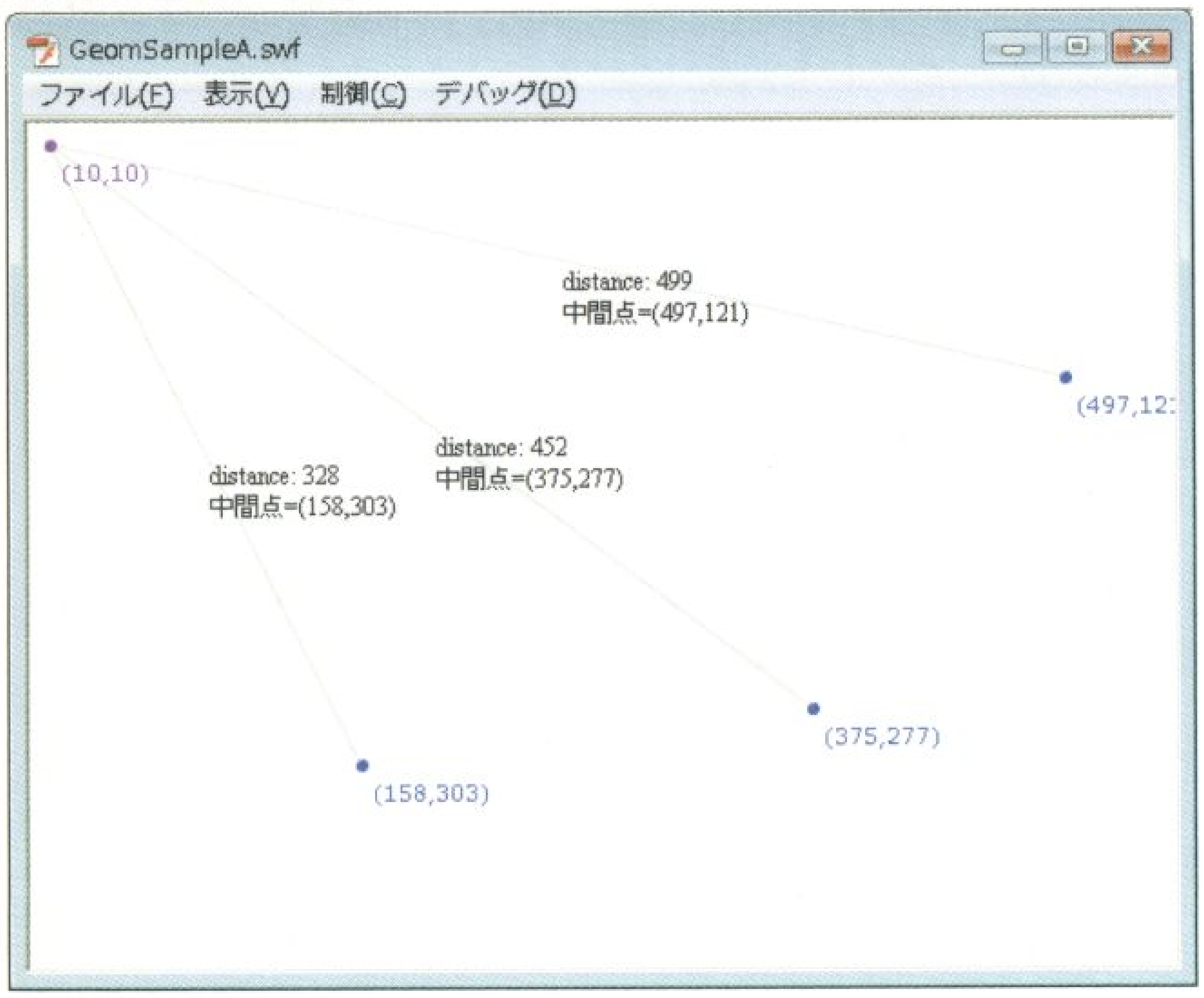
Pointクラスの主なメソッド概要

目的	メソッド名	メソッドの概要
オブジェクトのコピーを作成する	clone():Point	現在のPointオブジェクトのコピーを作成します。
別のPointオブジェクトとの距離を求める	distance(pt1:Point, pt2:Point):Number	パラメータで渡した2つのPointオブジェクトの距離を算出します。これはstaticメソッドです。

Pointクラスの主なメソッド概要（続き）

目的	メソッド名	メソッドの概要
別のPointオブジェクトと等しいか調べる	equals(toCompare:Point):Boolean	現在のオブジェクトがパラメータで渡されたPointオブジェクトと等しい座標かを調べます。
2つのPointオブジェクトの間に位置する座標を求める	interpolate(pt1:Point, pt2:Point, f:Number):Point	パラメータで指定した2つのPointオブジェクトの間に位置する座標を返します。fは補間レベルで、1ならpt1、0ならpt2を返します。これはstaticメソッドです。

SAMPLE



SAMPLE CODE

Pointクラスメソッドの使用 GeomSampleA.fla

```
import flash.display.Sprite
//基点とするPointオブジェクト
var startPoint:Point = new Point(10, 10);  ①
//マウスダウンのイベントハンドラ
this.stage.addEventListener(MouseEvent.MOUSE_DOWN, showPoint );
function showPoint(event:MouseEvent):void {
  var pt:Point = new Point( event.stageX, event.stageY );
  //ライブラリのmyPointクラスのインスタンスを作成して配置
  var obj :DisplayObject = this.addChild( new myPoint() );
  obj.x = pt.x;                                                  ②
  obj.y = pt.y;
  myPoint(obj).showPoint(  this as Sprite, startPoint  );
  //基点との中間点を算出
```

```
  var ppt:Point = Point.interpolate( startPoint, pt, 0.5 ); ———③
  var tf:TextField = new TextField();
  tf.multiline = true;
  //基点との距離を算出
  tf.text = "distance: " + int( Point.distance( startPoint, pt ) )
  + "¥n中間点=(" + pt.x + "," + pt.y + ")";
  tf.x = ppt.x;
  tf.y = ppt.y;                                                ④
  addChild( tf );
}
//基点を設定
function init():void {
  this.point1.x = startPoint.x;
  this.point1.y = startPoint.y;
  this.point1.showPoint( null, null );
}
init();
```


Pointクラスを使用して2つの座標の中間点と距離を算出するサンプルです。基点とする座標(10,10)は①でPointオブジェクトを作成しています。このPointオブジェクトと、マウスでクリックして配置されるムービークリップ(ライブラリ内のmyPoint)との中間点と距離をテキストフィールドで表示します。マウスをクリックしたときのムービークリップの配置は、②で行っています。新しいPointと基点との中間点は③、距離④でPointクラスのstaticメソッドを呼び出して算出しています。

ライブラリ内のムービークリップ(myPoint)

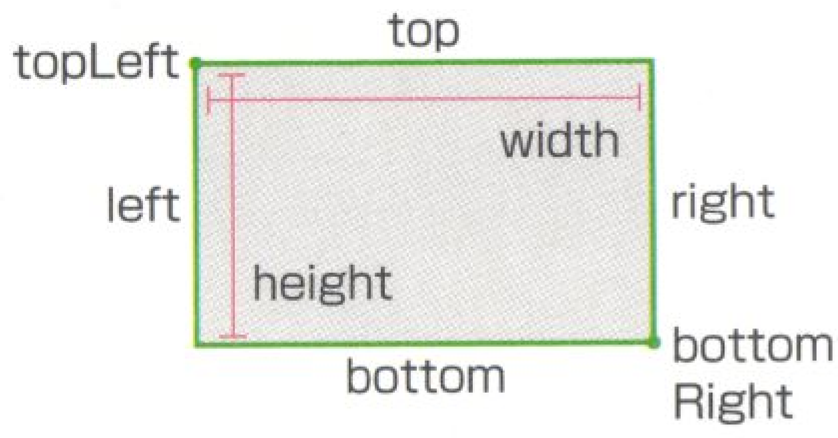
```
var startPoint:Point = new Point();
//自分の座標を表示して基点から線を引く関数
function showPoint( sprite:Sprite,  pt:Point ):void {
  point_txt.text = "(" + x + "," + y + ")";
  if ( sprite == null ) return;
  sprite.graphics.lineStyle( 1, 0xDDDDDD );
  sprite.graphics.moveTo( pt.x, pt.y );
  sprite.graphics.lineTo( x, y );
}
```

マウスでクリックした場所に点と座標を描くためのムービークリップです。わかりやすいように基点から線を引くようにしています。

7-9-2 Rectangleクラスを使う

Rectangleクラスは平面上の矩形（四角形）を表し、関連するメソッドを提供します。2次元の平面座標系ではステージと同様の矩形領域を扱う場合が多く、その座標とサイズを表すためにRectangleクラスオブジェクトが使用されます。オブジェクトにすることにより、矩形を表すデータをパラメータとして使用したり、クラスが提供するメソッドを使用することができます。

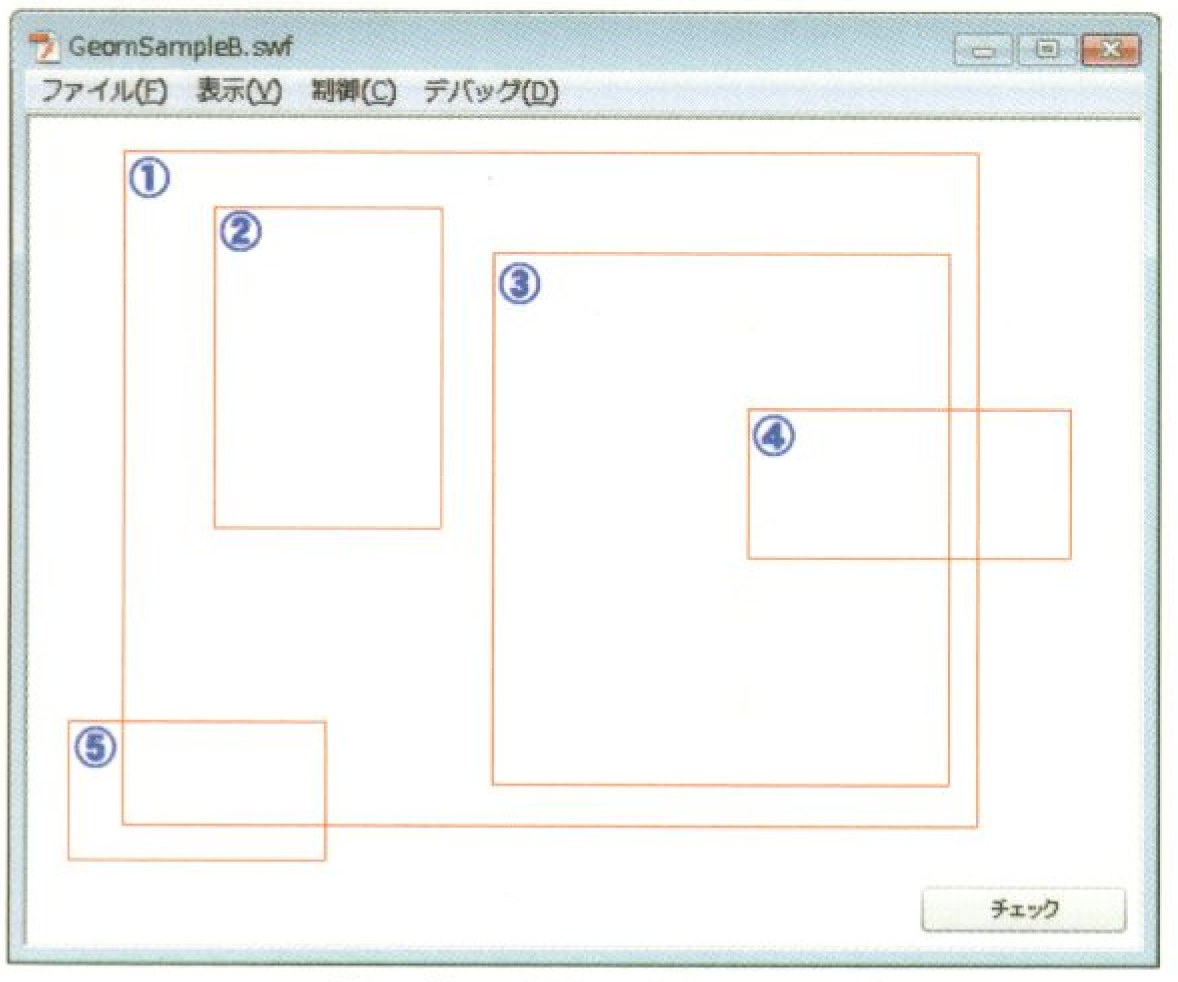
Rectangleクラスの主なプロパティ概要

<table>
<tr><th>目的</th><th>プロパティ名</th><th>プロパティの概要</th></tr>
<tr><td>矩形の高さ/幅を取得/設定する</td><td>height : Number
width : Number</td><td rowspan="7">それぞれのプロパティはx座標、またはy座標の値を表します。topLeftとbottomRightはx、y座標を持つPointオブジェクトです。

</td></tr>
<tr><td>左辺のx座標を取得/設定する</td><td>left : Number
x : Number</td></tr>
<tr><td>右辺のx座標を取得/設定する</td><td>right : Number</td></tr>
<tr><td>上辺のy座標を取得/設定する</td><td>top : Number
y : Number</td></tr>
<tr><td>下辺のy座標を取得/設定する</td><td>bottom : Number</td></tr>
<tr><td>左上角の座標を取得/設定する</td><td>topLeft : Point</td></tr>
<tr><td>右下角の座標を取得/設定する</td><td>bottomRight : Point</td></tr>
</table>

Rectangleクラスの主なメソッド概要

目的	メソッド名	メソッドの概要
オブジェクトのコピーを作成する	clone() :Rectangle	現在のRectangleオブジェクトとx、y、width、およびheightの各プロパティの値が等しい新しいRectangleオブジェクトを返します。
ある座標が矩形内にあるか調べる	containsPoint (point:Point) :Boolean	pointで指定された座標が現在のオブジェクト矩形領域内にあるかどうかを判別します。
別のRectangleオブジェクトが自オブジェクト内にあるか調べる	containsRect (rect:Rectangle) :Boolean	rectで指定されたRectangleオブジェクトが現在のオブジェクト矩形領域内にあるかどうかを判別します。
別のRectangleオブジェクトと自オブジェクトのどちらも含む矩形を表すRectangleオブジェクトを作成する	union (toUnion:Rectangle) :Rectangle	toUnionで指定されたRectangleオブジェクトと現在のオブジェクトを加算したRectangleオブジェクトを返します。

SAMPLE



※①～⑤の番号は実行時には表示されません。

SAMPLE CODE

Rectangleクラスを拡張するクラスの使用

GeomSampleB.fla

```
//カスタムクラスのインポート
import myPackage.MyRectangle;
import flash.geom.Rectangle;

//開始点と終了点を示すPointオブジェクト変数
var pt1:Point, pt2:Point;

//マウスのイベントハンドラ登録
this.stage.addEventListener(MouseEvent.MOUSE_DOWN, setStart);
this.stage.addEventListener(MouseEvent.MOUSE_UP, setEnd);

//マウスダウンの座標を開始点としてPointオブジェクトを作成       ─┐
function setStart(event:MouseEvent):void {                      ├─①
  pt1 = new Point( event.stageX, event.stageY );                 │
}                                                               ─┘

//マウスアップの座標を終了点としてPointオブジェクトを作成
function setEnd(event:MouseEvent):void {                        ─┐
  if ( pt1 == null ) return;                                     │
  pt2 = new Point( event.stageX, event.stageY );                 ├─②
  if ( pt2.equals( pt1 ) ) return;                               │
  //Rerctangleを拡張したカスタムクラスのオブジェクトを作成        │
```

```
  var rect:MyRectangle = new MyRectangle( pt1, pt2 );

  //シェイプを描画
  rect.drawMe( this, 1, 0xFF0000 );                    ②
  rects.push( rect );
}

//作成した四角形を格納する配列 ──────────────────────── ③
var rects:Array = new Array();
b1_btn.addEventListener( MouseEvent.CLICK, checkRects );
function checkRects(e:MouseEvent):void {
  if ( rects.length < 2 ) {
    trace( "比較する四角形がありません。" );
    return;
  }
  for ( var i:int = 1 ; i < rects.length ; i++ ) {
    //containRect()メソッドで2番目以降の四角形が1番目の四角形に含まれるか調べます
    if ( Rectangle( rects[0] ).containsRect( Rectangle( rects[i] ) )
    ) {
      trace( ( i + 1 ) + "番目の四角形が1番目の四角形の内側にあります。" );
    }
  }
}
```

このサンプルでは、Rectangleクラスを拡張するカスタムクラス(MyRectangle)を作成して使用します。プログラムの処理としては、マウスダウンで四角形の左上座標、ドラッグしてマウスアップした座標を四角形の右下となるようにMyRectangleオブジェクトを作成してステージ上に描画します。マウスダウンの処理は①マウスアップの処理とMyRectangleオブジェクトの作成は②で行っています。③以降はオプションの処理で、最初に描いた四角形の中に2番目以降のものが含まれるかをcontainsRect()メソッドで調べる処理を行っています。

myPackage.MyRectangle.as

```
package myPackage{
  import flash.display.Shape;
  import flash.geom.Rectangle;
  import flash.geom.Point;
  import flash.display.DisplayObjectContainer;
```

```
  import flash.events.MouseEvent;

  public class MyRectangle extends Rectangle  {
    var s:Shape;
    //コンストラクタ(pt1:左上座標、pt2:右下座標)                        ┐
    public function MyRectangle( pt1:Point, pt2:Point) {                │
    //Rectangleクラスのコンストラクタにパラメータを渡してオブジェクトを構築 ├─④
    super(  pt1.x, pt1.y, (pt2.x - pt1.x), (pt2.y - pt1.y) );           │
    }                                                                   ┘
    //自分が持っているRectangleオブジェクトの矩形を描画する関数              ⑤
    public function drawMe( stg:DisplayObjectContainer, thickness:
    Number, color:uint ):Shape {
      s = new Shape();
      s.graphics.lineStyle( thickness, color );
      s.graphics.moveTo( 0, 0 );
      s.graphics.lineTo( this.width, 0 );
      s.graphics.lineTo( this.width, this.height );
      s.graphics.lineTo( 0, this.height );
      s.graphics.lineTo( 0, 0 );
      stg.addChild( s );
      s.x = this.x;
      s.y = this.y;
      return s;
    }
  }
}
```

Rectangleクラスを拡張するMyRectangleクラスです。MyRectangleクラスでは、④のコンストラクタで矩形の左上と右下の座標を示す2つのPointオブジェクトを受け取り、スーパークラスのRectangleコンストラクタに渡します。⑤のdrawMe()メソッドでは、DisplayObjectContainerオブジェクトを受け取り、そこに自分が持つRectangleデータに基づいてシェイプを描画します。

▶ 実行結果（トレース出力：「チェック」ボタンを押した場合）

```
2番目の四角形が1番目の四角形の内側にあります。
3番目の四角形が1番目の四角形の内側にあります。
```

7-9-3 Matrixクラスで移動、回転、拡大/縮小を行う

表示オブジェクトのx、y、rotation、witdh、height、scaleX、scaleYプロパティなどを直接変更することで移動やサイズ変更などの処理を行えますが、Matrixクラスを使用するとメソッドを使用したり、複数の変更をまとめて設定してからオブジェクトに適用することができます。表示オブジェクトにMatrixオブジェクトを適用するには、tramsformプロパティで取得できるTransformオブジェクトのmatrixプロパティに作成したMatrixオブジェクトを設定します。Matrixクラスでは、プロパティの概要にあるような行列に変換のための数値を格納します。この行列は変換マトリックスと呼ばれ、行列内のプロパティはそれぞれ目的が決まっているため、直接編集することも可能です。Matrixクラスを使用して表示オブジェクトの変更を行う一般的な手順は次のようになります。

Matrixクラスで表示オブジェクトの座標変換を行う手順

No.	作業内容	使用する主なクラス
1	対象のオブジェクトのtrasformプロパティでTransformオブジェクトを取得し、そのオブジェクトのmatrixプロパティで得られるMatrixオブジェクトを変数に格納する	Transform（参照のみ） Matrix
2	格納したMatrixオブジェクトに目的の変換を適用する	Matrix
3	変換を適用したMatrixオブジェクトを再び対象オブジェクトのtransform.matrixプロパティに設定する	—

Matrixクラスの主なプロパティ概要

目的	プロパティ名	プロパティの概要
x軸方向の拡大/縮小、回転のパラメータを設定	a :Number	変換マトリックス内で目的の操作を行うためのパラメータです。 $\begin{bmatrix} a & b & t_x \\ c & d & t_y \\ 0 & 0 & 1 \end{bmatrix}$ （0と1の数値が書かれた場所は、常にその値であることを表します）
y軸方向の拡大/縮小、回転のパラメータを設定	d :Number	
x軸方向の回転、傾斜のパラメータを設定	b :Number	
y軸方向の回転、傾斜のパラメータを設定	c :Number	
x軸方向の移動距離を設定	tx :Number	
y軸方向の移動距離を設定	ty :Number	

Matrixクラスの主なメソッド概要

目的	メソッド名	メソッドの概要
オブジェクトのコピーを作成する	`clone():Rectangle`	現在のMatrixオブジェクトと同じ変換マトリックスを持つ新しいRectangleオブジェクトを返します。
回転、拡大/縮小、平行移動の変換をまとめて設定する	`createBox( scaleX:Number, scaleY:Number, rotation:Number = 0, tx:Number = 0, ty:Number = 0 ):void`	回転、拡大/縮小、平行移動に関する変換を引数で指定すると、オブジェクト内の変換マトリックスに適用されます。
適用するグラデーションスタイルを作成する	`createGradientBox(width:Number, height:Number, rotation:Number = 0, tx:Number=0, ty:Number=0 ):void`	GraphicsクラスのbeginGradientFill()メソッドおよびlineGradientStyle()メソッドで使用する特定のスタイルを作成します。
回転変換を適用する	`rotate( angle:Number ):void`	オブジェクトの変換マトリックスに回転変換を適用します。
拡大/縮小変換を適用する	`scale( sx:Number, sy:Number ):void`	オブジェクトの変換マトリックスに行列に拡大/縮小を適用します。
平行移動を適用する	`translate( dx:Number, dy:Number ):void`	オブジェクトの変換マトリックスに平行移動を適用します。
現在のマトリックスをあるPointオブジェクトに適用した結果を得る	`transformPoint( point:Point ):Point`	pointで与えられるオブジェクトに現在の変換を適用して新しいPointオブジェクトを返します。

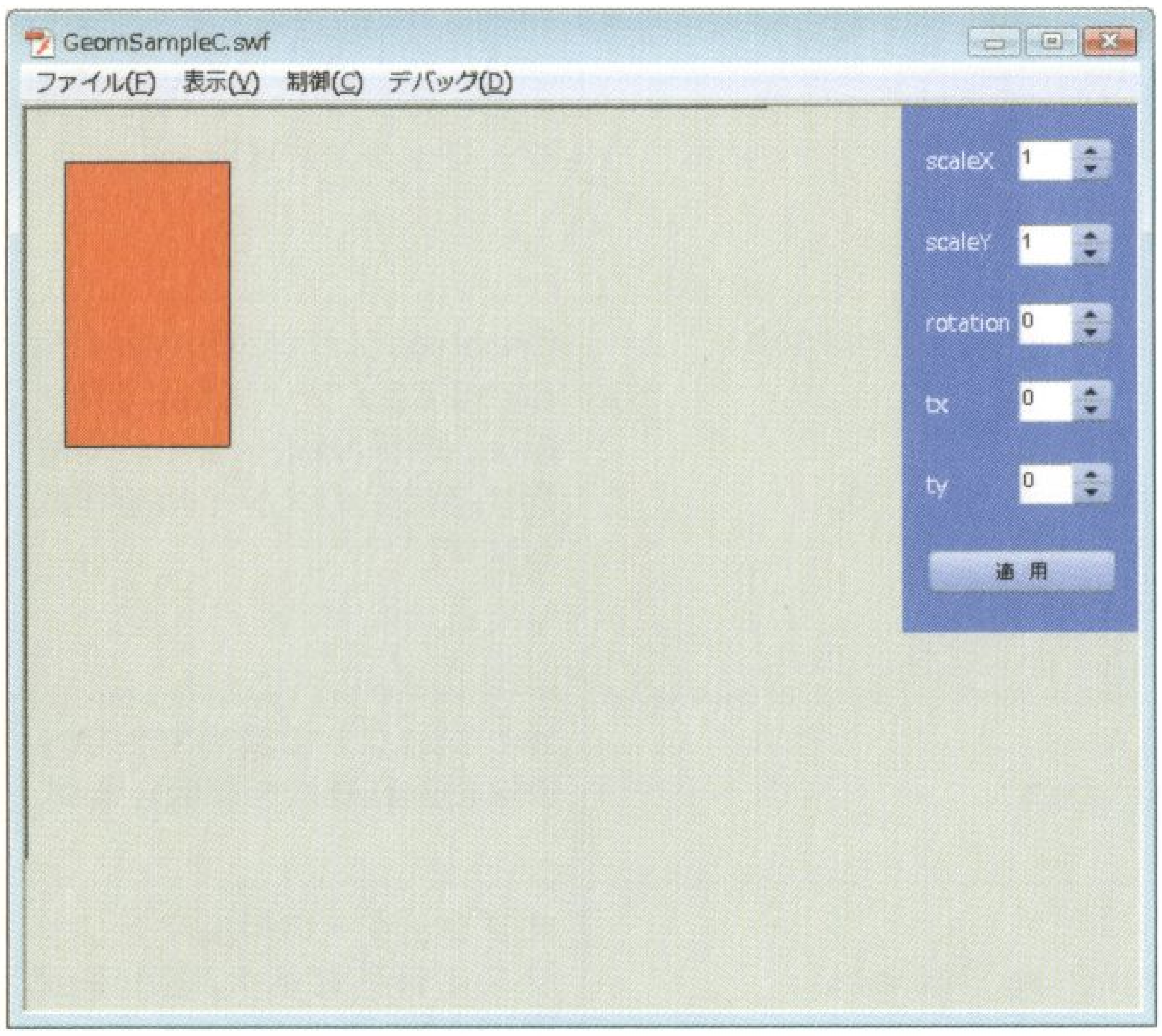
Matrixクラスの移動、回転、拡大/縮小に関するメソッド詳細

メソッド	メソッドの詳細
`rotate( angle:Number ):void`	angleで指定した角度で表示オブジェクトを回転させます。回転は親コンテナの基準点を中心とする回転になります。角度(angle)はラジアン表記であるため、0～±360°から次のような計算で変換して渡す必要があります。 ラジアン = 2 * Math.PI * ((0～±360の角度) / 360); 90度の回転を加える場合、次のような計算をします。 angle = 2 * Math.PI * (90 / 360)
`scale( sx:Number, sy:Number ):void`	sx,syでそれぞれ水平方向、垂直方向の拡大、縮小を行います。変換はローカル座標系の拡大/縮小となり、表示オブジェクトが基準点(0,0)から離れている場合、x、y座標も変化します。sx、syは乗数で指定するので、1.5を指定すると1.5倍の拡大になります。

9
3

Matrixクラスの移動、回転、拡大/縮小に関するメソッド詳細(つづき)

メソッド	メソッドの詳細
translate(dx:Number, dy:Number):void	dx、dyで指定したピクセル分、オブジェクトを現在の位置から水平、垂直移動します。



マウスをドラッグすると矩形を描くことができます。

SAMPLE CODE

Matrixオブジェクトによる回転、拡大/縮小、移動を行う

GeomSampleC.fla

```
import flash.geom.Matrix;
//カスタムクラスのインポート
import myPackage.CustomEvent;
import myPackage.MyRectangle2;
import flash.geom.Rectangle;
//開始点と終了点を示すPointオブジェクト変数
var pt1:Point, pt2:Point;
//Matrixを適用する表示オブジェクト
var shap:Shape;
//マウスのイベントハンドラ登録
this.stage.addEventListener(MouseEvent.MOUSE_DOWN, setStart);
this.stage.addEventListener(MouseEvent.MOUSE_UP, setEnd);
this.addEventListener("matrixData", getMatrix);
function getMatrix(e:CustomEvent):void {
```

```
    if ( shap == null ) return;
    //現在のMatrixオブジェクトを取得して座標変換を行う
    var mtx:Matrix = shap.transform.matrix;
    //拡大/縮小変換
    if ( Number(e.data.scaleX) != 1 || Number(e.data.scaleY) != 1 )
      mtx.scale(  Number(e.data.scaleX), Number(e.data.scaleY) );
    //回転変換
    if ( Number(e.data.rotation) != 0 )
      mtx.rotate( Number(e.data.rotation) );
    //平行移動
    if ( Number(e.data.tx) != 0 || Number(e.data.ty)  != 0 )
      mtx.translate( Number(e.data.tx), Number(e.data.ty) );
    //Matrixオブジェクトを適用
    shap.transform.matrix = mtx;
}                                                              ①
//マウスダウンの座標を開始点としてPointオブジェクトを作成
function setStart(event:MouseEvent):void {
    //パラメータ設定用のパネル上なら描画を行わない
    if ( event.target.name  == "panel_mc" ) return;
    pt1 = new Point( event.stageX, event.stageY );
}
//マウスアップの座標を終了点としてPointオブジェクトを作成
function setEnd(event:MouseEvent):void {
    if ( pt1 == null ) return;
    pt2 = new Point( event.stageX, event.stageY );
    if ( pt2.equals( pt1 ) ) return;
    //Rerctangleを拡張したカスタムクラスのオブジェクトを作成
    var rect:MyRectangle2 = new MyRectangle2( pt1, pt2 );
    //シェイプを描画
    shap = rect.fillMe( this, 1, 0x000000, 0xFF4400 );
    //イベントリスナー登録の削除
    this.stage.removeEventListener(MouseEvent.MOUSE_DOWN, setStart);
    this.stage.removeEventListener(MouseEvent.MOUSE_UP, setEnd);
}
```

画面右上の変換パラメータを設定するパネルは、ムービークリップになっています。下記のソースをご覧ください。このソースでは、パネルから送られるパラメータを①のコードでステージ上に作成されたShapeオブジェクトに手順のとおり適用します。②以降はマウスでShapeオブジェクトを描画するための手順で、Matrixとは関連はありません。

ライブラリ内のムービークリップ(matrixPanel)

```
import flash.geom.Matrix;
import myPackage.CustomEvent;
this.apply_btn.addEventListener(MouseEvent.CLICK, clickHandler );
function clickHandler(event:MouseEvent):void {
  var obj:Object = new Object();
  obj.scaleX = this.scaleX_ns.value;
  obj.scaleY = this.scaleY_ns.value;
  obj.rotation = 2* Math.PI * ( this.rotation_ns.value / 360 );
  obj.tx = this.tx_ns.value;
  obj.ty = this.ty_ns.value;
  var ev:CustomEvent = new CustomEvent("matrixData");
  ev.data = obj;
  parent.dispatchEvent( ev ); ――――――――――――――――――― ③
}
```

配置されたNumericStepperオブジェクトの入力値をカスタムイベントオブジェクトに格納してディスパッチします。イベントタイプは"matrixData"です。メインタイムラインソースの③でこのイベントをリスンする設定を行っています。

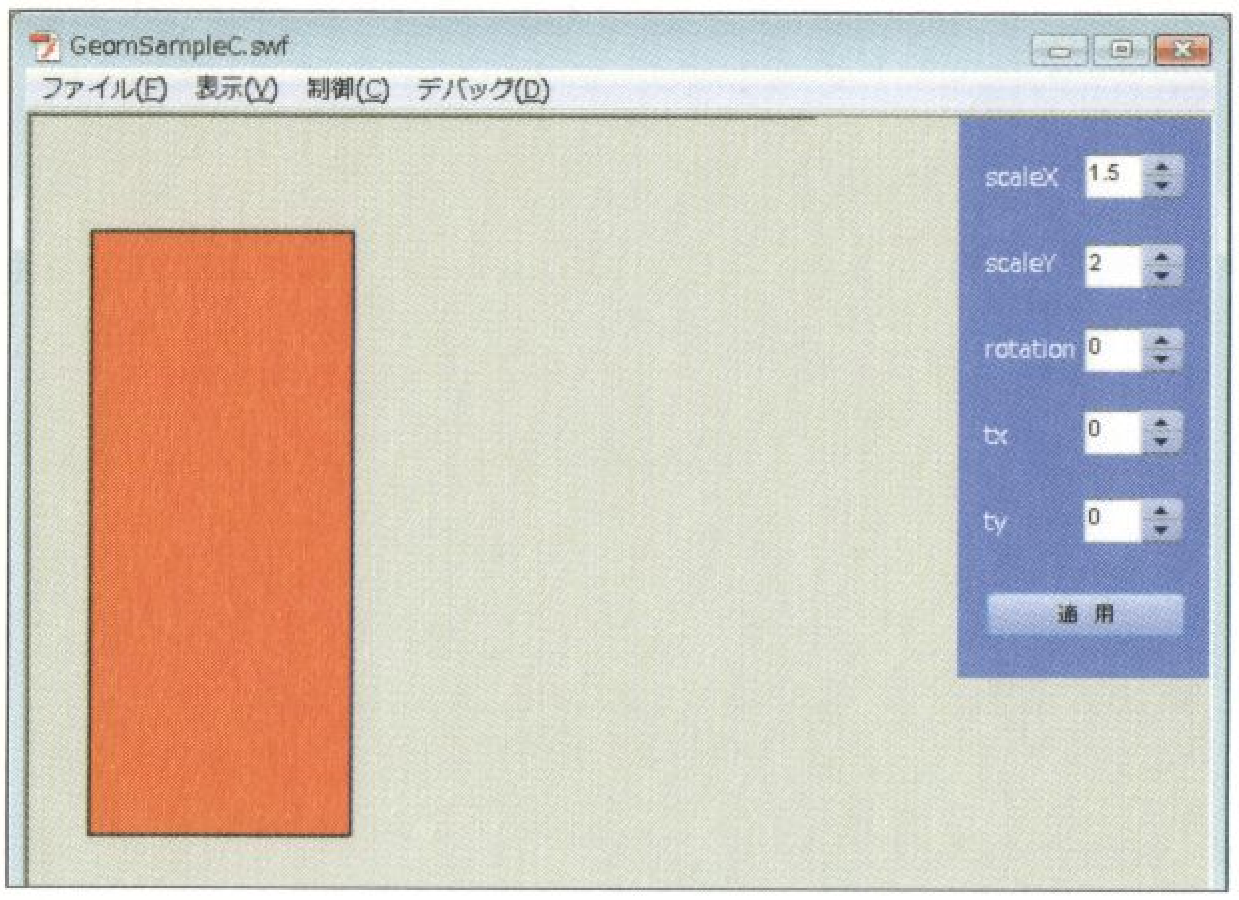
myPackage.MyRectangle2.as

```
package myPackage{
import flash.display.Shape;
import flash.geom.Rectangle;
import flash.geom.Point;
import flash.display.DisplayObjectContainer;
import flash.events.MouseEvent;
public class MyRectangle2 extends Rectangle  {
  var s:Shape;
  //コンストラクタ(pt1:左上座標、pt2:右下座標)
  public function MyRectangle2( pt1:Point, pt2:Point) {
  //Rectangleクラスのコンストラクタにパラメータを渡してオブジェクトを構築
  super(  pt1.x, pt1.y, (pt2.x - pt1.x), (pt2.y - pt1.y) );
  }
  //自分が持っているRectangleオブジェクトの矩形を描画する関数
  public function drawMe( stg:DisplayObjectContainer, tickness:
  Number, color:uint ):Shape {
    s = new Shape();
    s.graphics.lineStyle( thickness, color );
```

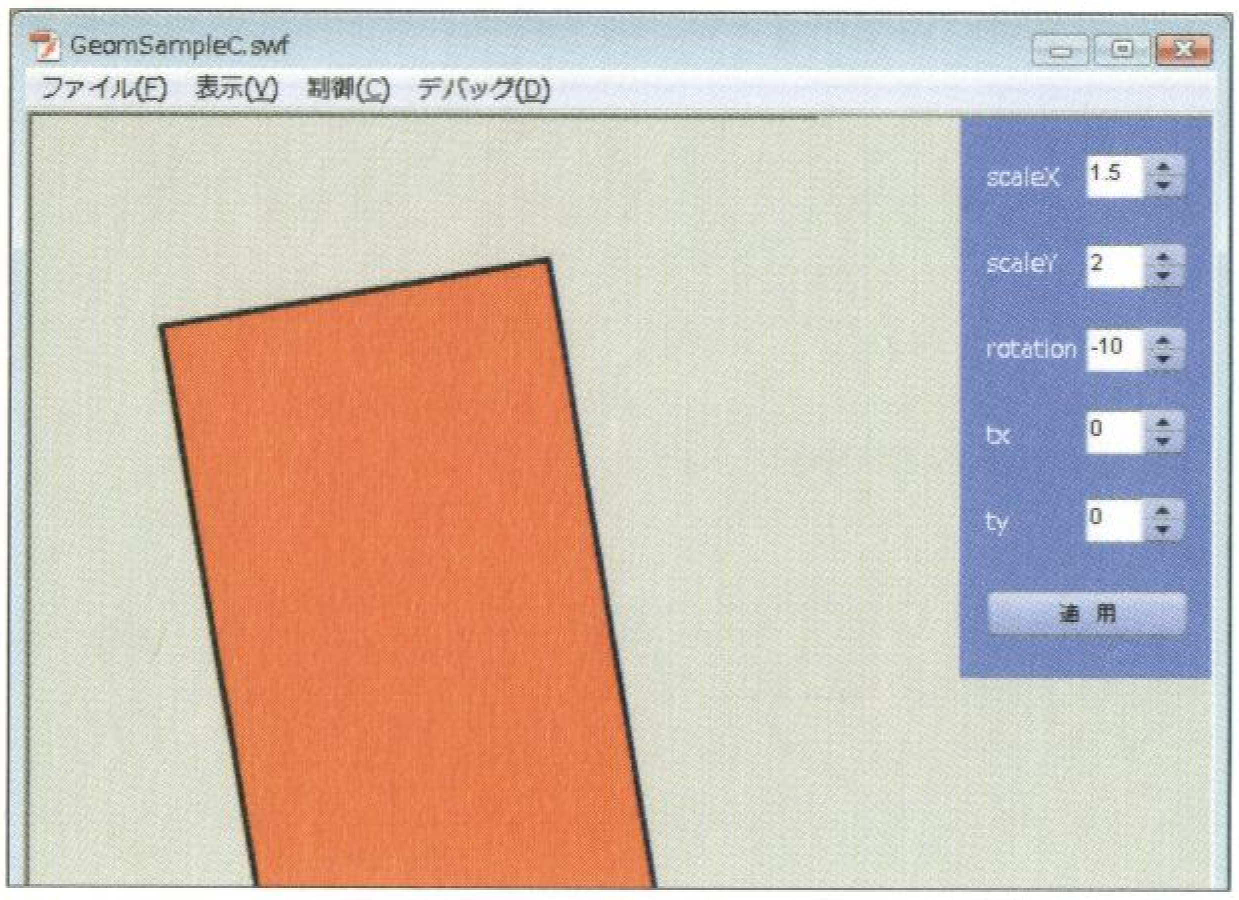
```
    s.graphics.moveTo( 0, 0 );
    s.graphics.lineTo( this.width, 0 );
    s.graphics.lineTo( this.width, this.height );
    s.graphics.lineTo( 0, this.height );
    s.graphics.lineTo( 0, 0 );
    stg.addChild( s );
    s.x = this.x;
    s.y = this.y;
    return s;
  }
  //自分が持っているRectangleオブジェクトの矩形を描画(fill)する関数
  public function fillMe( stg:DisplayObjectContainer, thickness:
  Number, color:uint, fillcolor:uint ):Shape {
    s = new Shape();
    s.graphics.beginFill( fillcolor );
    s.graphics.lineStyle( tickness, color );
    s.graphics.moveTo( 0, 0 );
    s.graphics.lineTo( this.width, 0 );
    s.graphics.lineTo( this.width, this.height );
    s.graphics.lineTo( 0, this.height );
    s.graphics.lineTo( 0, 0 );
    stg.addChild( s );
    s.x = this.x;
    s.y = this.y;
    return s;
    }
  }
}
```


シェイプ描画を行うRectangleを拡張したクラスは 7-9-2 「Rectangleクラスを使う」のサンプルで使用したものに微細な追加を行ったものです。fillMe()メソッドはbeginFill()メソッドを呼び出して矩形内部に塗りを加えたのみです。

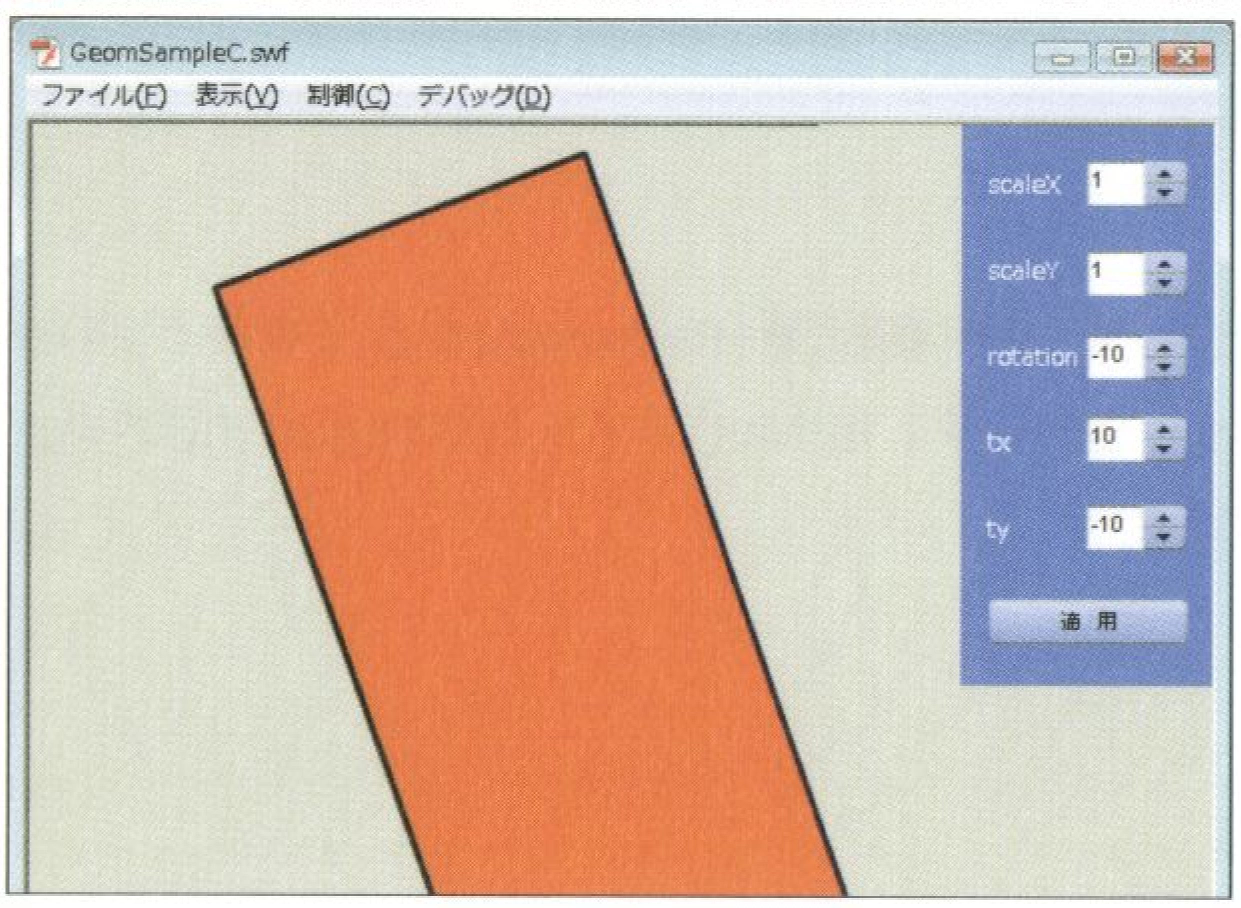
■ **scaleX=1.5、scaleY=2を適用**



■ **同じscaleにrotation=-10を追加して適用**



■ **scaleX、scaleY=1に戻し、rotation=-10にtx=10、ty=-10を追加して適用**



7-10 3D

7-10-1 3D関連機能をサポートするクラス

Flash Player10とデスクトップ実行環境であるAIRの1.5以降のバージョンでは、すべての表示オブジェクトにzプロパティがあります。このzプロパティは3D座標系でオブジェクトの配置に奥行きを与えたり、ユーザの視点から距離を持たせるために使用されます。

表示オブジェクトを3D表示するには、zプロパティに数値を適用したり、3Dをサポートする回転、拡大/縮小プロパティを使用することで3Dアニメーションが実現できます。3D関連機能をサポートするActionScript3.0クラスには、下記の表のものがあります。

3D関連機能をサポートするActionScript3.0クラス

パッケージ	クラス	内容
flash.display	DisplayObject	ActionScript3.0の表示オブジェクトはzプロパティと、3D空間の表示オブジェクトを操作するための回転、拡大/縮小プロパティがあります。local3DToGlobal( )メソッドは3Dジオメトリを2D平面に投影する方法を提供します。
flash.geom	PerspectiveProjection	遠近法による表示オブジェクトの変形をサポートします。3Dジオメトリを2D平面ビューにマッピングするためのパラメータが制御されます。
	Matrix3D	3Dジオメトリの回転、拡大/縮小、平行移動を行う複雑な変換を簡単に行える方法を提供します。
	Vector3D	3Dポイントを管理するためのデータ構造を表します。ベクター数式もサポートします。

DisplayObjectクラスの3Dをサポートするプロパティ詳細

プロパティ	プロパティの詳細
x y z :Number	x、y、z座標により、表示オブジェクトインスタンスの3D空間内の位置を決定します。プロパティを変化させることで、3D空間内オブジェクトを移動します。
rotationX rotationY rotationZ :Number	x軸、y軸、z軸を中心に、表示オブジェクトインスタンスを3D空間で回転します。プロパティの値は回転角度を表します。時計回りに回転させる場合は0～180、反時計回りに回転させる場合は0～-180の値を指定します。この範囲を超える値は、360を加算または減算して、範囲内に収まる値になるように調整されます。
scaleX scaleY scaleZ :Number	表示オブジェクトインスタンスのx、y、z方向のスケール(伸縮)をパーセンテージで表します。scaleZを変化させることで、3D空間内のオブジェクトの奥行きスケールを変更できます。
transform :Transform	3Dプロパティの設定により自動生成されるMatrix3Dオブジェクトへの参照を含むTransformオブジェクトにアクセスできます。Matrix3Dオブジェクトは3D空間内の表示オブジェクトの表現方法を管理します。

DisplayObjectクラスの3Dをサポートするメソッド詳細

メソッド	メソッドの詳細
globalToLocal3D(point:Point):Vector3D	pointで示されるステージ座標(グローバル座標)の2次元のポイントを、3次元の表示オブジェクトのローカル座標内の3次元のポイントに変換します。戻り値はその3次元ポイントを持つVector3Dオブジェクトです。
local3DToGlobal(point3d:Vector3D):Point	3次元の表示オブジェクトのローカル座標の3次元のポイントをステージ座標(グローバル座標)の2次元のポイントに変換します。 Graphicsクラスなどの3D座標をサポートしないクラスで3D図形の描画を行う場合、3D=>2Dの座標変換が必要になります。このメソッドで3D座標を変換することにより、複雑な計算を独自に行う必要はなく、3D座標のみを用意して奥行きのある図形を描画することができます。 3D座標を持つVector3Dオブジェクトを作成してパラメータ(point3d)に渡すことで、2次元の座標を持つPointオブジェクトが返されます。

PerspectiveProjectionクラスの主なプロパティ概要

目的	プロパティ名	プロパティの概要
遠近法の消失点を設定する	projectionCenter:Point	projectionCenterプロパティは、遠近法投影の消失点を表します。
遠近法の視野を角度で指定する	fieldOfView:Number	遠近法投影の強度を決定する0～180度の角度を指定します。
遠近法の焦点の長さを取得(設定)する	focalLength:Number	視点の原点(0,0,0)とz軸上の表示オブジェクトの位置との間の距離を表します。

Matrix3Dクラスの主なプロパティ概要

目的	プロパティ名	プロパティの概要
対象オブジェクトの3D座標を調べる	position:Vector3D	3D処理を行っているオブジェクトの現在の3D座標位置を保持するVector3Dインスタンスを参照します。

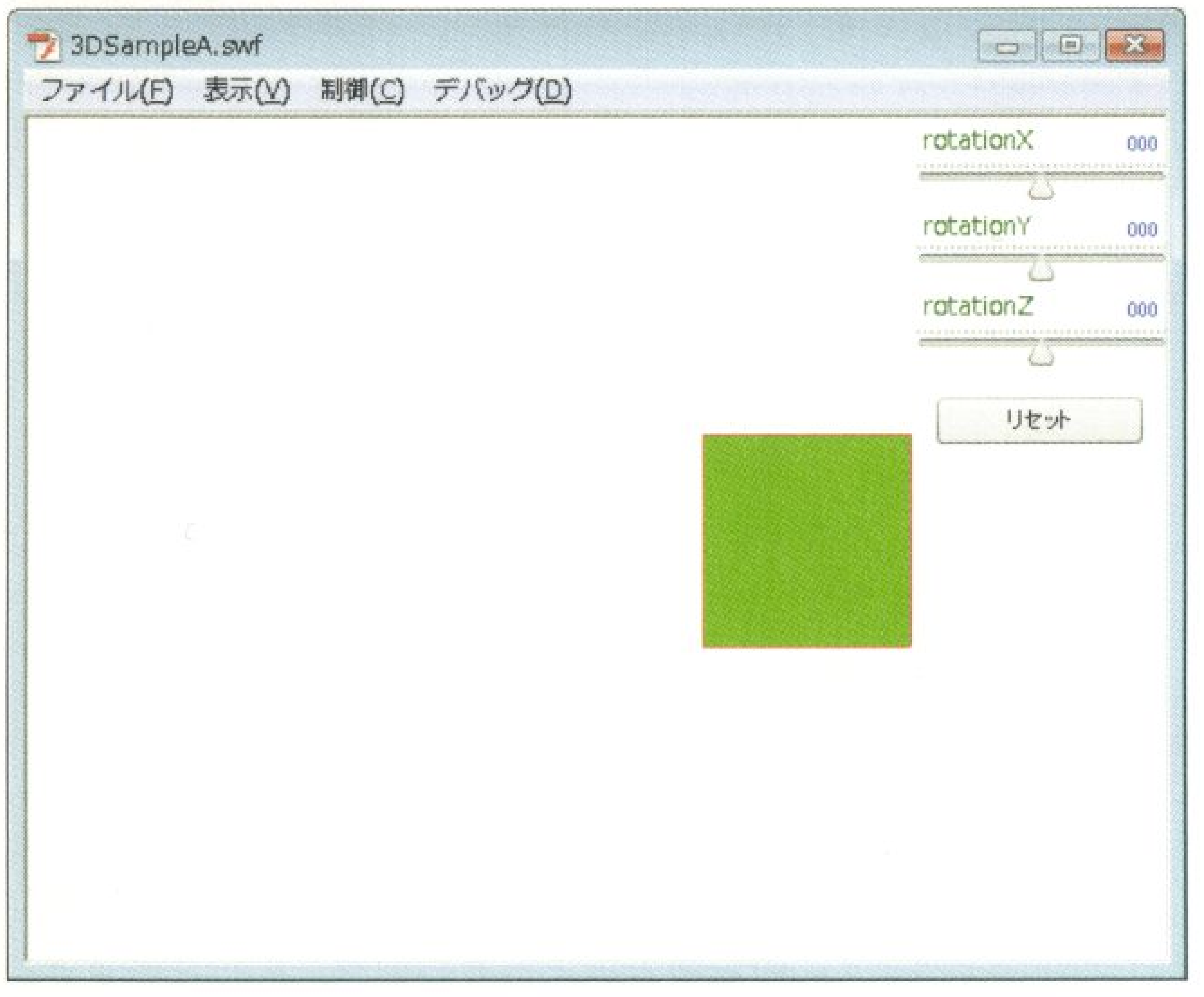
Matrix3Dクラスの主なメソッド概要

目的	メソッド名	メソッドの概要
x/y/z軸での回転を加える	appendRotation(degrees:Number, axis:Vector3D, pivotPoint:Vector3D = null):void prependRotation(degrees:Number, axis:Vector3D, pivotPoint:Vector3D = null):void	指定した角度と軸を中心とした回転をMatrix3Dオブジェクトに追加します。そのオブジェクトを表示オブジェクトに適用すると、appendRotation()メソッドの場合、他の変換の後に回転が実行されます。prependRotation()メソッドの場合、他の変換の前に回転が実行されます。

Matrix3Dクラスの主なメソッド概要（続き）

目的	メソッド名	メソッドの概要
x/y/z軸方向に拡大/縮小を行う	appendScale(xScale:Number, yScale:Number, zScale:Number):void prependScale(xScale:Number, yScale:Number, zScale:Number):void	指定した軸上の拡大/縮小をMatrix3Dオブジェクトに追加します。そのオブジェクトを表示オブジェクトに適用すると、appendScale()メソッドの場合、他の変換の後に拡大/縮小が実行されます。prependScale()メソッドの場合、ほかの変換の前に拡大/縮小が実行されます。
x/y/z軸方向の位置を移動する	appendTranslation(x:Number, y:Number, z:Number):void prependTranslation(x:Number, y:Number, z:Number):void	指定した軸上の位置の変更をMatrix3Dオブジェクトに追加します。そのオブジェクトを表示オブジェクトに適用すると、appendScale()メソッドの場合、他の変換の後に位置の変更が実行されます。prependScale()メソッドの場合、他の変換の前に位置の変更が実行されます。
Matrix3Dオブジェクトのクローン(コピー)を作成する	clone():Matrix3D	現在のMatrix3Dオブジェクトのコピーを返します。
逆の変換を行う反転行列を作成する	invert():Boolean	現在オブジェクトが持っているマトリックスを反転したオブジェクトを返します。
別に作成したMatrix3Dオブジェクトを追加する	append(lhs:Matrix3D):void prepend(rhs:Matrix3D):void	別のMatrix3Dオブジェクトを現在のオブジェクトに追加します。追加後に表示オブジェクトに適用すると、append()メソッドの場合、他の変換の後に追加した内容が実行されます。prepend()メソッドの場合、他の変換の前に追加した内容が実行されます。

SAMPLE



SAMPLE CODE

シンプルな3Dアニメーション

3DSampleA.fla

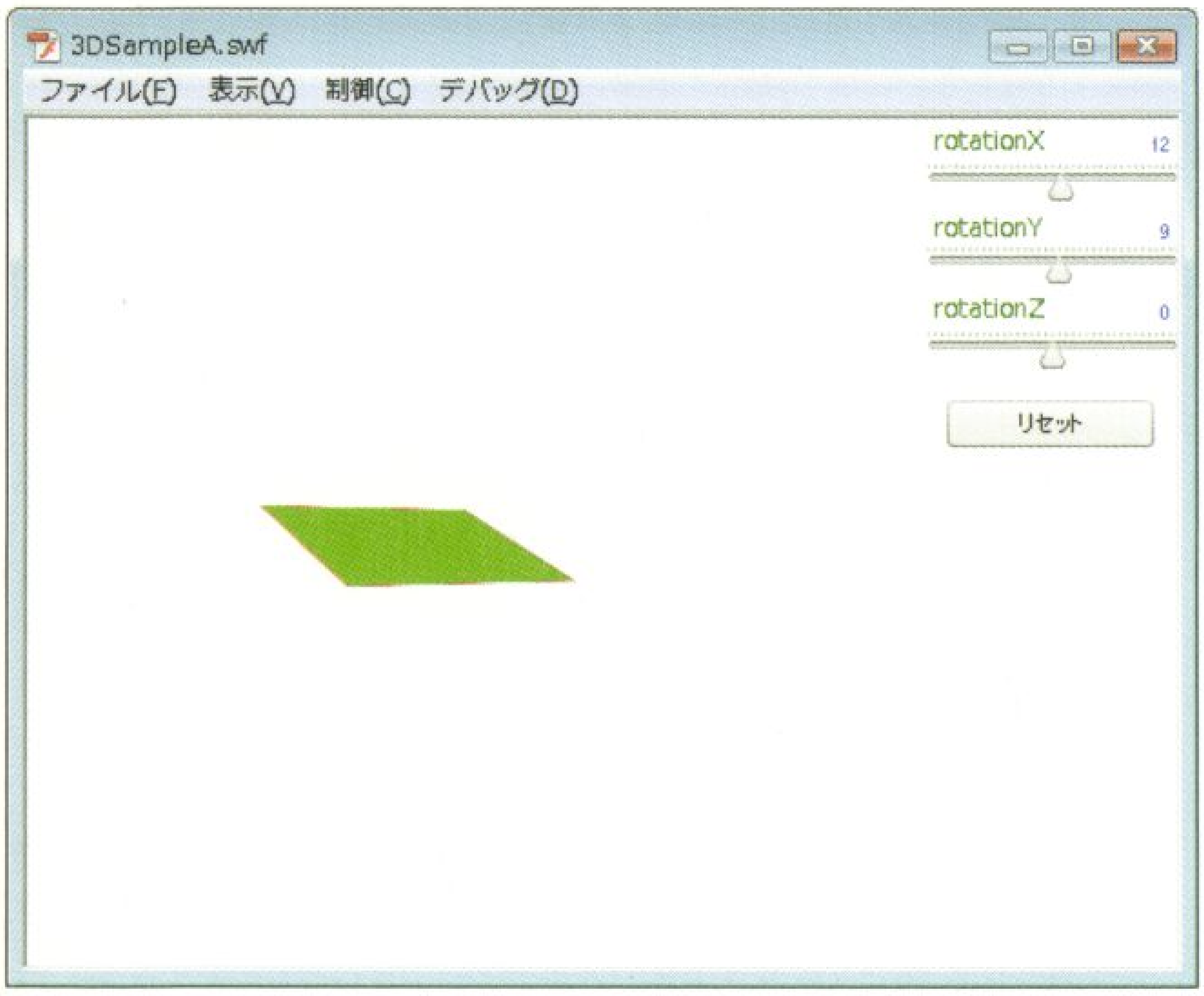
```
import flash.display.Shape;
import flash.display.Sprite;
import flash.events.Event;
//Rotationの基準とする変数
var xr:int=0;
var yr:int=0;
var zr:int=0;
//図形オブジェクト
var s:Shape;
function init():void {
   //対象の図形を描画
   drawShape();
   //フレームレートのタイミングで処理を行うためのイベントハンドラ登録
   this.addEventListener(Event.ENTER_FRAME, enterFrameFunctions);
   //rotation
   rotationX_ns.addEventListener( Event.CHANGE, setRotation );
   rotationY_ns.addEventListener( Event.CHANGE, setRotation );
   rotationZ_ns.addEventListener( Event.CHANGE, setRotation );
   //リセットボタン
   reset_btn.addEventListener( MouseEvent.CLICK, resetShape );
}
```

①

```
function drawShape():void {
  //すでに描画していたら、クリア
  if ( s != null ) s.graphics.clear(); ――③
  s = new Shape();
  s.graphics.lineStyle( 1, 0xff0000, 0.8 );
  s.graphics.beginFill( 0x00ff00 );                  ――②
  s.graphics.drawRect(50, -50, 100, 100);
  s.x=stage.stageWidth/2;
  s.y=stage.stageHeight/2;
  this.addChild(s);
}
function resetShape(event:MouseEvent):void {
  //NumericStepperの値をリセット
  rotationX_ns.value = 0;
  rotationY_ns.value = 0;
  rotationZ_ns.value = 0;
  //ここで渡すeventはダミー
  setRotation(event);
  //シェイプを再描画
  drawShape();
}
function enterFrameFunction(event:Event):void {
  //変数の値により、3Dパラメータを更新
  s.rotationX += xr;                                 ――④
  s.rotationY += yr;
  s.rotationZ += zr;
}
function setRotation(event:Event):void {
  //NumericStepperから値を取り出して変数に代入
  xr=rotationX_ns.value;
  rotationX_txt.text=rotationX_ns.value.toString();
  yr=rotationY_ns.value;
  rotationY_txt.text=rotationY_ns.value.toString();
  zr=rotationZ_ns.value;
  rotationZ_txt.text=rotationZ_ns.value.toString();
}
init();
```

このサンプルでは、3D操作の基本となるRotationプロパティの値を変化させます。表示オブジェクトが持つRotationX、RotationY、RotationZはそれぞれx軸、 y 軸、 z 軸を基準にした回転角度を表します。このプロパティを連続的に変化させることで、動きのある3Dアニメーションを作成することができます。

②のdrawShape()関数で対象のShapeオブジェクトを作成して表示しています。回転角度を表す変数は①で宣言している変数です。RotationX/Y/ZプロパティはNumber型ですが、ここでは整数値で問題ないため、int型で作成しています。回転のプロパティを変化させるのは、④のenterFrameFunction()関数で行っています。回転を加えると表示オブジェクトの形状はその回転状態を表す形に変化し、軸も変化してしまうので、リセットして最初の状態に戻すボタンをつけました。ボタンを押したときの処理は②のdrawShape()関数を呼び出していますが、この関数はinit()初期処理でも描画のために使用します。そのため、③で行っているようにすでに描画しているとs変数に値が入っているので、そのときはGraphicsクラスのclear()メソッドを呼び出して古いシェイプを消しています。



rotationX/rotationYの変化値を与えると、x軸で回転しながらy軸で回転を続けます。

7-10-2 PerspectiveProjection オブジェクトで遠近法を使う

PerspectiveProjectionクラスは、3D 空間で表示オブジェクトに基本的な遠近法による変形を適用することができます。遠近法による投影を行わない場合もデフォルトの設定値によるPerspectiveProjectionオブジェクトが使用されます。デフォルトでは、Flash Playerによってステージの中央が消失点として自動的に設定されます。消失点はzプロパティを増大させてユーザから遠ざけるときにすべてのオブジェクトが収束する点です。

PerspectiveProjection クラスの遠近法投影のためのプロパティ詳細

プロパティ	プロパティの詳細
projectionCenter :Point	遠近法投影の消失点をPointオブジェクトで表します。こステージの左上隅を消失点とする場合、オブジェクトが持つ座標値を(0,0)にします。表示オブジェクトをユーザから遠い方へ移動させる場合、そのオブジェクトは設定した消失点に向かって収束し、最終的に消失します。ルートに作成されるデフォルトのPerspectiveProjectionオブジェクトはステージの中央を表すprojectionCenter値を持ちます。
fieldOfView :Number	遠近法投影の強度を決定する1以上180未満の角度を指定します。3次元の視野の角度を変化させることにより、3D表示オブジェクトがユーザに近付くほど大きく表示され、ユーザから遠ざかるほど小さく表示されるようにできます。 値が大きいほど、z軸に沿って移動する表示オブジェクトに適用される歪みが強くなります。値が小さい場合、移動するオブジェクトの拡大/縮小が少なくなり、オブジェクトは空間内で若干後方に移動するように見えるのみになります。値が大きい場合、移動に伴う歪みが大きくなり、より大幅に移動したように見えます。最大値の180度に近い値を指定すると、極端な魚眼カメラレンズ効果が生成されます。 ルートに作成されるデフォルトのPerspectiveProjectionオブジェクトは55というfieldOfView値を持ちます。
focalLength :Number	視点の原点(0,0,0)とz軸上の表示オブジェクトの位置との間の距離を表します。この距離は焦点の長さと呼ばれます。focalLengthは、表示オブジェクトが移動するときの遠近法変換中に動的に再計算されますが、値を設定することもできます。 ルートに作成されるデフォルトのPerspectiveProjectionオブジェクトは次の式で算出されるfocalLength値を持ちます。stageWidth/ 2 * (cos(fieldOfView/2) / sin(fieldOfView/2))

PerspectiveProjection クラスのメソッド詳細

メソッド	メソッドの詳細
toMatrix3D( ):Matrix3D	表示オブジェクトの基になるMatrix3Dオブジェクトを返します。

SAMPLE



※初期状態（消失点:ステージ中央、fieldOfView=55）

SAMPLE CODE

PerspectiveProjectionオブジェクトのプロパティを確認する

3DSampleB.fla

```
import flash.display.Shape;
import flash.geom.Point;
import flash.events.Event;
import flash.geom.PerspectiveProjection;
//シェイプ用の変数
var s:Shape;
function init() {
  //perspectiveProjection.fieldOfViewを1(最小に近い値)にしておく
  root.transform.perspectiveProjection.fieldOfView = 1;
  //1000個のシェイプを作成
  for (var i:int = 0; i < 1000; i++) {
    s = new Shape();
    //色をランダムに
    s.graphics.beginFill(Math.random() * 0xFFFFFF);
    s.graphics.drawRect( -5, -5, 5, 40 );
    //x座標をステージ幅を最大とするランダム値に
    s.x=Math.random()*stage.stageWidth;
    //y座標はステージの中央
    s.y=stage.stageHeight/2;
```

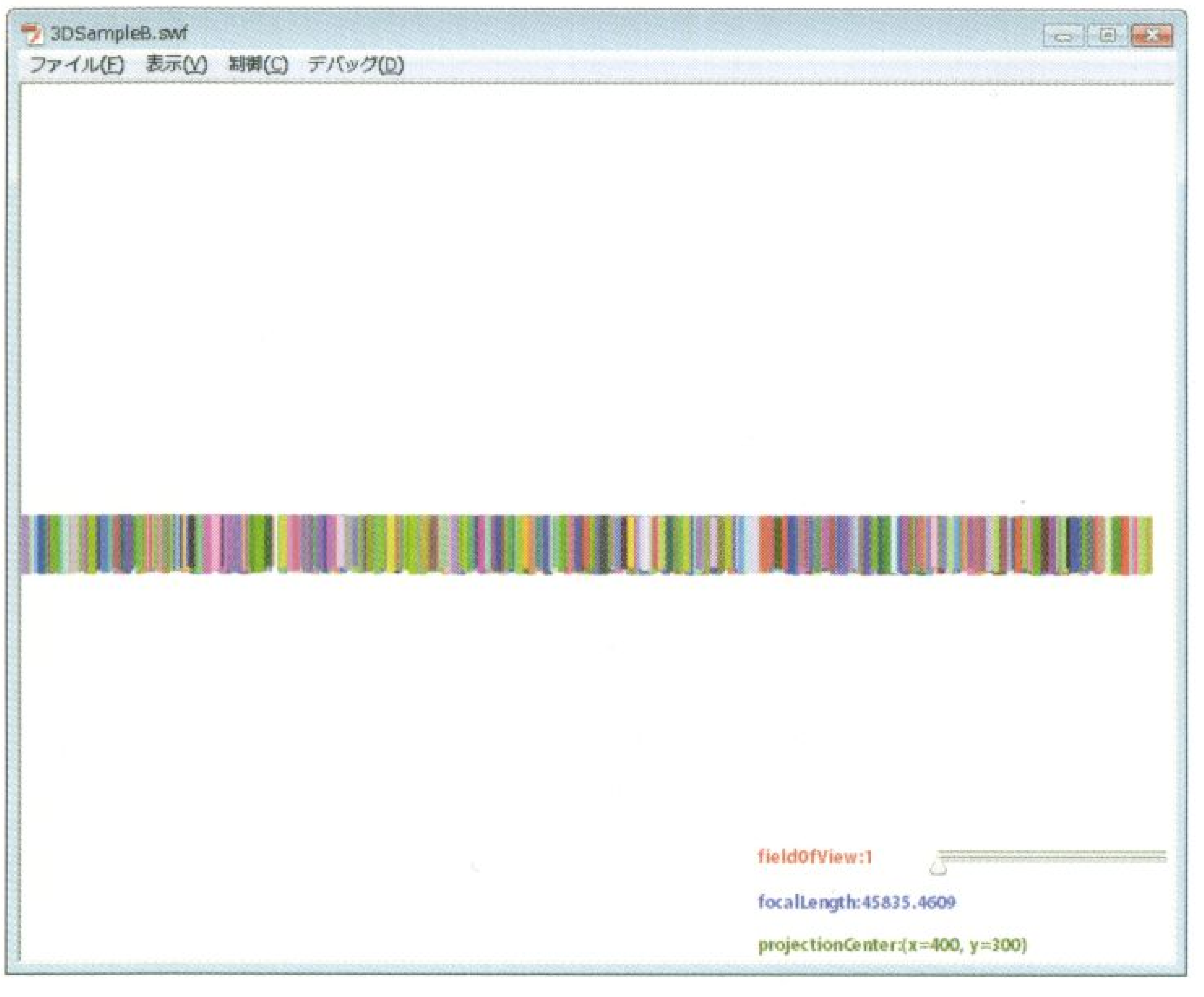
①

```
    //z座標を0～8800未満のランダム値に
    s.z=Math.random()*8800;
    addChild( s );
  }
  //マウスクリックハンドラの登録
  stage.addEventListener(MouseEvent.CLICK, changePP );          ①
  showInfo();
}
function changePP(event:MouseEvent):void {
  //マウスをクリックした座標をperspectiveProjection.projectionCenterにする
  setRoot( event.stageX, event.stageY ); ――――――――――――――――― ②
  showInfo();
}
function setRoot(x:Number, y:Number):void {
  //perspectiveProjection.projectionCenteを変更
  root.transform.perspectiveProjection.projectionCenter=new
  Point(x,y);
}
//スライダーのイベントハンドラ登録
fv.addEventListener( Event.CHANGE, setFieldOfView );
//perspectiveProjection.fieldOfViewを変更する関数
function setFieldOfView( e:Event ):void {
  root.transform.perspectiveProjection.fieldOfView = fv.value; ―― ③
  showInfo();
}
function showInfo():void {
  fv_txt.text = "fieldOfView:" + fv.value;
  fl_txt.text = "focalLength:" + root.transform.
  perspectiveProjection.focalLength;
  pc_txt.text = "projectionCenter:" + root.transform.
  perspectiveProjection.projectionCenter.toString();// as Point;
}
init();
```

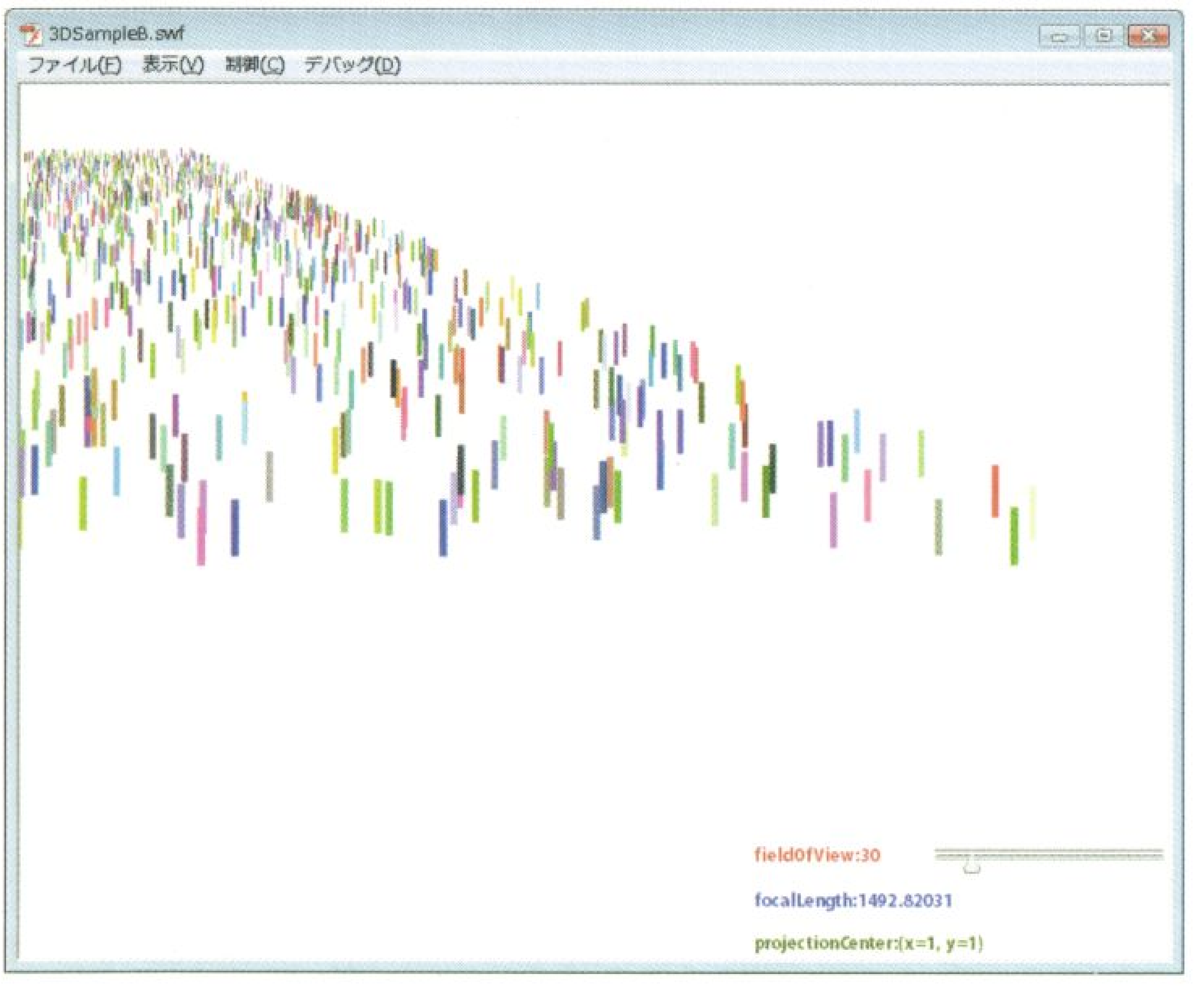
①のinit()関数で1000個のシェイプをランダムな場所に作成します。z座標もランダムにするため、消失点に向かって遠くなってゆくイメージを作成しています。ただし、コードをシンプルでわかりやすくするためにオブジェクトの重ね合わせの調整をしていないため、よく見ると奥にあるはずのシェイプが前面に描画されたりしています。②でマウスをクリックした座標を

perspectiveProjection.projectionCenterにしています。これにより、消失点の位置が変わり、「もっとも遠い地点」が変化します。ステージ右下に配置したスライダーコントロールでperspectiveProjection.fieldOfViewの値を変更し、結果を確認できます。変更は③で行っています。

■ 消失点:ステージ中央、fieldOfView=1



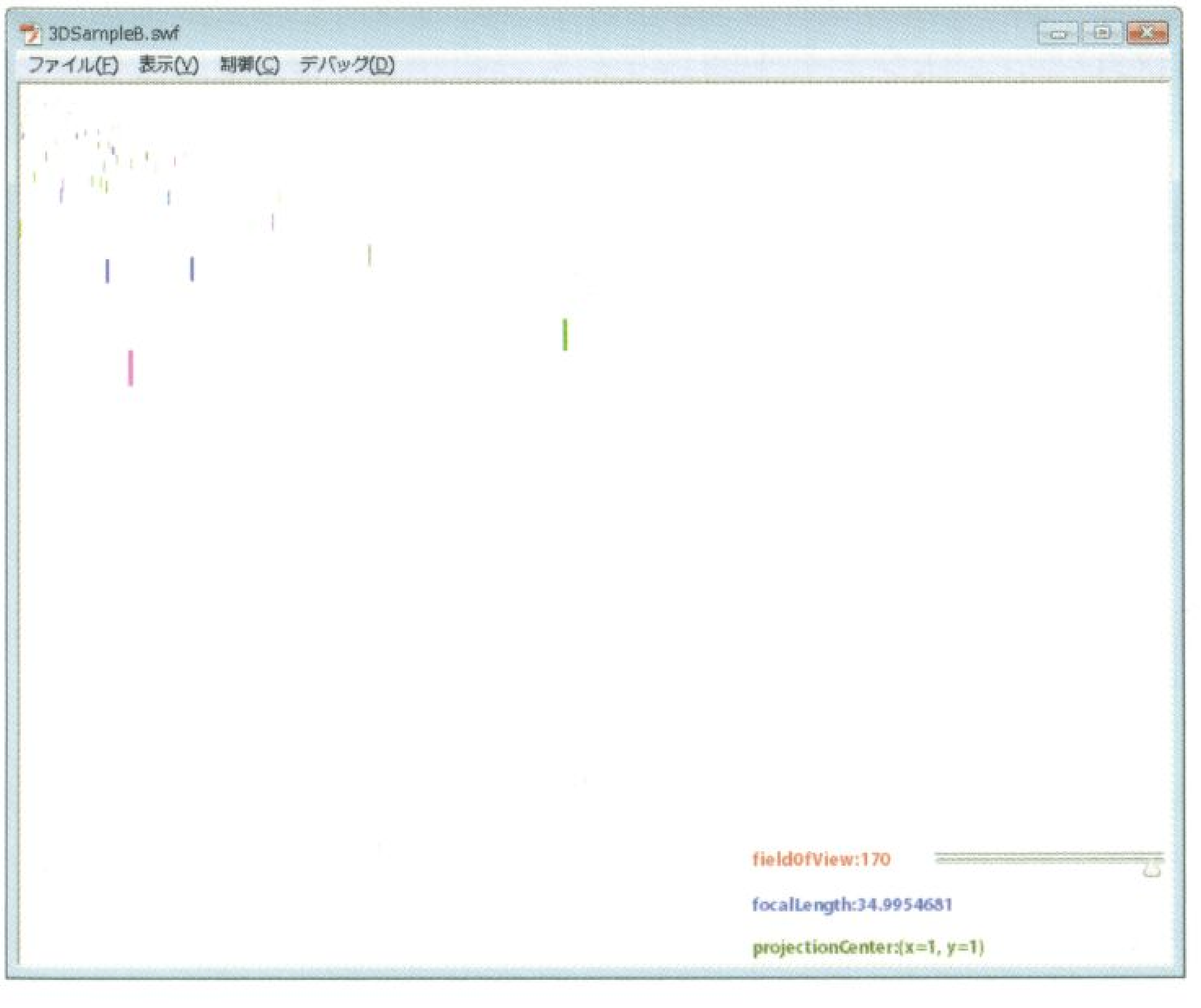
■ 消失点:ステージ左上(1,1)、fieldOfView=30



■ 消失点:ステージ左上(1,1)、fieldOfView=90



■ 消失点:ステージ左上(1,1)、fieldOfView=170



7-10-3 Matrix3Dオブジェクトを使う

Matrix3Dクラスに用意されたメソッドにより、オブジェクトの移動やサイズ変更、回転を行うための3D座標変換を行うことができます。変換は表示オブジェクトのプロパティやメソッドを使用して1つずつ行うことができますが、Matrix3Dオブジェクトにより複数の変換を1回で効率的に行うことができます。ただし、適用する変換の種類とパラメータがそれぞれ同じでも、個々の変換を適用する順序により最終結果が大きく異なるので注意が必要です。Matrix3Dオブジェクトは変換用のデータを下図のような4行x4列の行列で保持し、これは変換マトリックスと呼ばれます。ただし、このマトリックスの詳細を数学的に理解する必要はありません。表示オブジェクトにMatrix3Dオブジェクトによる変換を行う場合の基本手順として、まず表示オブジェクトが持つMatrix3Dオブジェクトを取得し、それに対して変換メソッドを直接実行する方法と、独自の変換パラメータを設定したMatrix3Dオブジェクトを作成し、適用（追加）する方法があります。表示オブジェクトがすでにMatrix3Dオブジェクトを持っている場合、そのオブジェクトはtransformプロパティで参照できるTransformオブジェクトのmatrix3Dプロパティで取得できますが、デフォルトでは作成されないため、下表のいずれかの操作により、あらかじめMatrix3Dオブジェクトを生成しておく必要があります。

■ Matrix3Dオブジェクトの変換マトリックス

方向　　　　　変換

$$
\begin{bmatrix}
scaleX & 0 & 0 & t_x \\
0 & scaleY & 0 & t_y \\
0 & 0 & scaleZ & t_z \\
0 & 0 & 0 & t_w
\end{bmatrix}
$$

x-座標　y-座標　z-座標

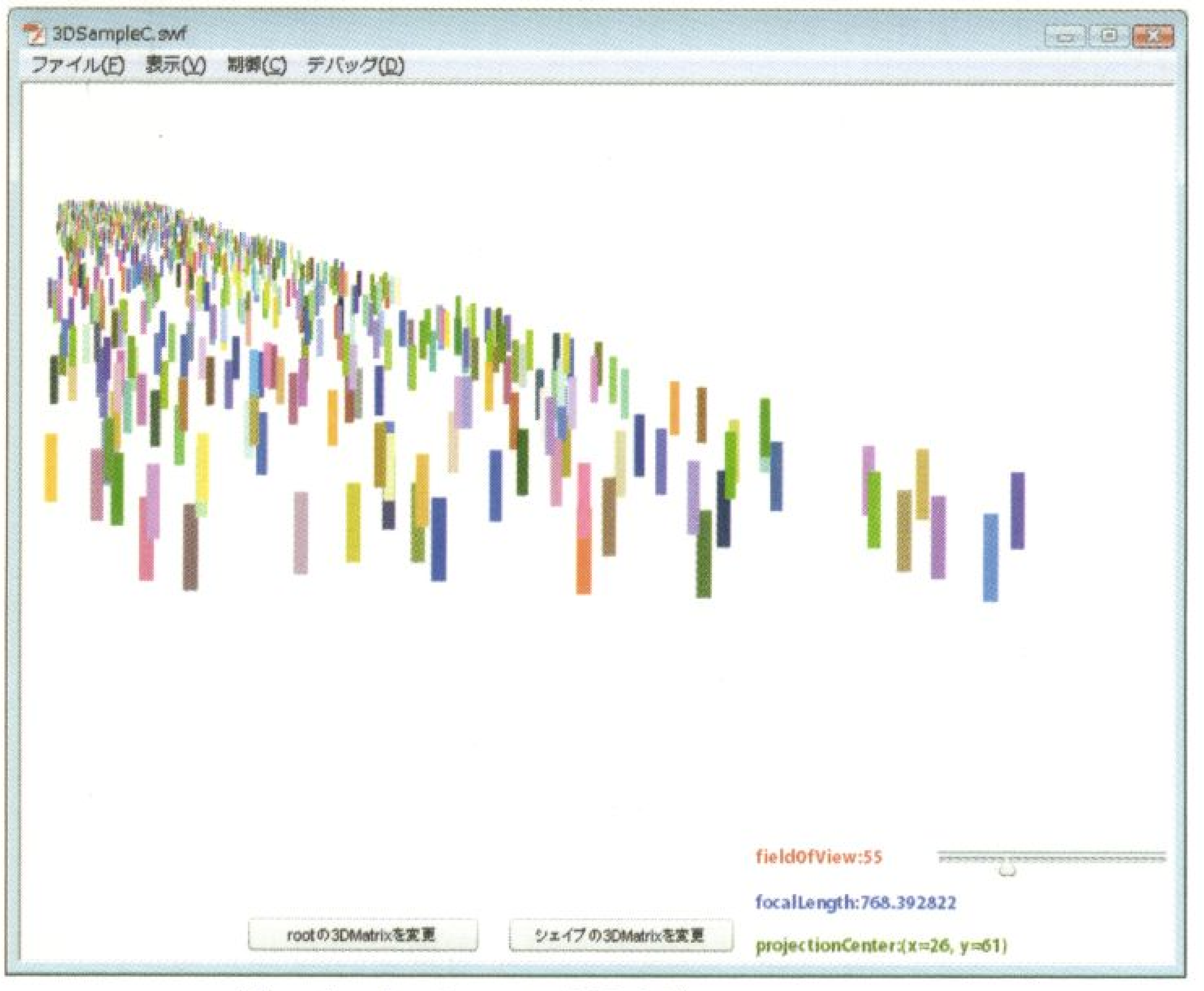
■ 表示オブジェクトのMatrix3Dオブジェクトを作成する方法

No.	処理内容
1	表示オブジェクトに対するMatrix3Dは3Dに関連するプロパティを設定することで自動的に生成されます。例えば、zプロパティに数値0を設定することで、オブジェクトの位置を変更しないでMatrix3Dオブジェクトを生成できます。
2	Matrix3Dインスタンスを明示的に生成します。コンストラクタにパラメータを渡して独自の変換を行うためのオブジェクトを作成できます。
3	ルート表示オブジェクトのperspectiveProjection.matrix3Dプロパティにより、ステージ上の3Dオブジェクトの表示を制御するMatrix3D オブジェクトを取得できます。

Matrix3Dクラスの主なメソッド詳細

メソッド名	メソッドの概要
appendRotation(degrees:Number, axis:Vector3D, pivotPoint:Vector3D = null):void prependRotation(degrees:Number, axis:Vector3D, pivotPoint:Vector3D = null):void	指定した角度と軸を中心とした回転をMatrix3Dオブジェクトに追加します。appendRotation()メソッドは他の変換の後、prependRotation()メソッドは他の変換の前に指定した回転が実行されます。 **degrees**：Number 適用する回転の角度です。 **axis**：Vector3D Vector3Dクラスの定数で回転の軸を指定します。 Vector3D.X_AXIS: x軸を表します。 Vector3D.Y_AXIS: y軸を表します。 Vector3D.Z_AXIS: z軸を表します。 pivotPoint:Vector3Dオブジェクト回転の中心を決める3D座標ポイントです。
appendScale(xScale:Number, yScale:Number, zScale:Number):void prependScale(xScale:Number, yScale:Number, zScale:Number):void	指定した軸上の拡大/縮小をMatrix3Dオブジェクトに追加します。appendScale()メソッドは他の変換の後、prependScale()メソッドは他の変換の前に拡大/縮小が実行されます。拡大/縮小は乗数、つまり掛け算の値で設定します。1.5の設定値は1.5倍の拡大になります。変更を行わない軸の値は1にしておきます。 xScale:Number x軸方向の拡大/縮小乗数です。 yScale:Number y軸方向の拡大/縮小乗数です。 zScale:Number z軸方向の拡大/縮小乗数です。
appendTranslation(x:Number, y:Number, z:Number):void prependTranslation(x:Number, y:Number, z:Number):void	指定した軸上の位置の変更をMatrix3Dオブジェクトに追加します。appendScale()メソッドはほかの変換の後、prependScale()メソッドはほかの変換の前に位置の変更が実行されます。位置の変更は増分、つまり現在の位置にプラスする数値を設定します。10の設定値は指定した軸方向に10移動します。位置を変更しない軸の値は0にしておきます。 x:Number x軸方向の位置の増分です。 y:Number y軸方向の位置の増分です。 z:Number z軸方向の位置の増分です。
clone():Matrix3D	現在のMatrix3Dオブジェクトのコピーを返します。既存のMatrix3Dオブジェクトに変換メソッドを実行すると、そのオブジェクトが保持しているマトリックスが変更されてしまいます。変換前のオリジナルマトリックスを保持しておきたいときは、clone()メソッドでコピーを作成してから変換を加えます。
invert():Boolean	現在オブジェクトが持っているマトリックスを反転したオブジェクトを返します。反転したマトリックスにより、行った変換を元に戻すことができます。
append(lhs:Matrix3D):void prepend(rhs:Matrix3D):void	別のMatrix3Dオブジェクトを乗算により、現在のオブジェクトに追加します。append()メソッドは他の変換の後に、prepend()メソッドは、他の変換の前に追加した内容が実行されます。

SAMPLE



※projectionCenter: (26,61)

SAMPLE CODE

Matrix3Dによる変換

3DSampleC.fla

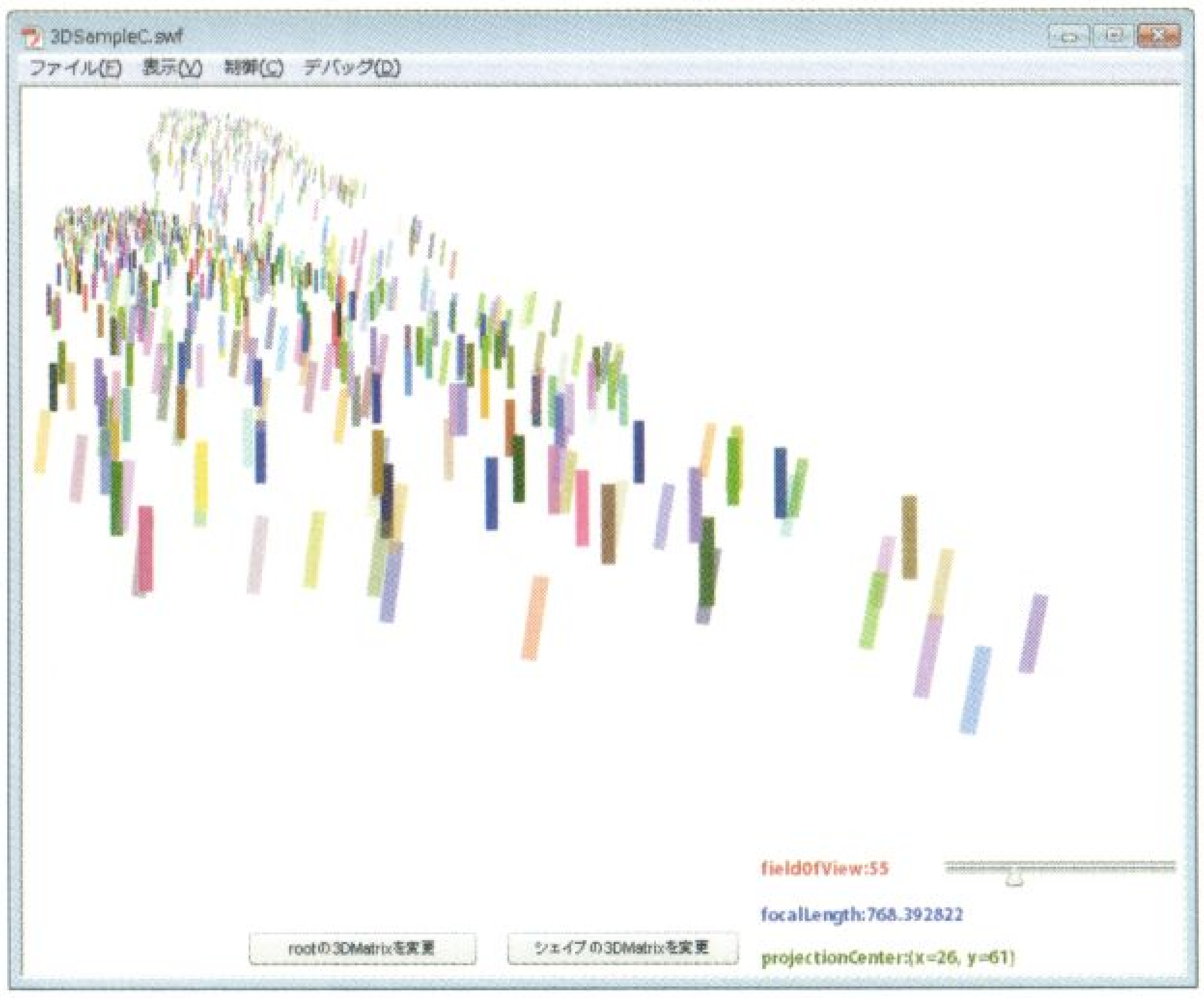
```
import flash.display.Shape;
import flash.geom.Point;
import flash.events.Event;
import flash.geom.PerspectiveProjection;
import flash.geom.Vector3D;
//シェイプ用の変数
var s:Shape;
var shapes:Array; ―――――――――――――――――――――――――――――― ①
var curShape:Shape;
//Matrix3Dオブジェクト用の変数
var mat3:Matrix3D;
//初期処理関数
function init() {
  shapes = new Array();
  //perspectiveProjection.fieldOfViewを1(最小に近い値)にしておく
  root.transform.perspectiveProjection.fieldOfView=55;
  //1000個のシェイプを作成
  for (var i:int = 0; i < 1000; i++) {
    s = new Shape();
    //色をランダムに
```

```
      s.graphics.beginFill(Math.random() * 0xFFFFFF);
      s.graphics.drawRect( -5, -5, 10, 60 );
      //x座標をステージ幅を最大とするランダム値に
      s.x=Math.random()*stage.stageWidth;
      //y座標はステージの中央
      s.y=stage.stageHeight/2;
      //z座標を0～8800未満のランダム値に
      s.z=Math.random()*8800;
      addChild( s );
      //シェイプを配列に格納
      shapes.push( s ); ―――――――――――――――――――――― ②
    }
    //マウスクリックハンドラの登録
    stage.addEventListener(MouseEvent.CLICK, changePP );
    //シェイプの3DMatrixを変更ボタンイベントハンドラ
    mat_btn.addEventListener(MouseEvent.CLICK, setMatrix3D);
    //rootの3DMatrixを変更ボタンイベントハンドラ
    matroot_btn.addEventListener(MouseEvent.CLICK, setMatrix3Droot);
    showInfo();
  }
  function changePP(event:MouseEvent):void {
    if (event.target==this.fv) {
      return;
    }
    if (event.target==this.mat_btn||event.target==this.matroot_btn) {
      return;
    }//マウスをクリックした座標をperspectiveProjection.projectionCenterにする
    setRoot( event.stageX, event.stageY );
    showInfo();
  }
  function setRoot(x:Number, y:Number):void {
    //perspectiveProjection.projectionCenterを変更
    root.transform.perspectiveProjection.projectionCenter=new Point
    (x,y);
  }
  //スライダーのイベントハンドラ登録
  fv.addEventListener( Event.CHANGE, setFieldOfView );
  //perspectiveProjection.fieldOfViewを変更する関数
```

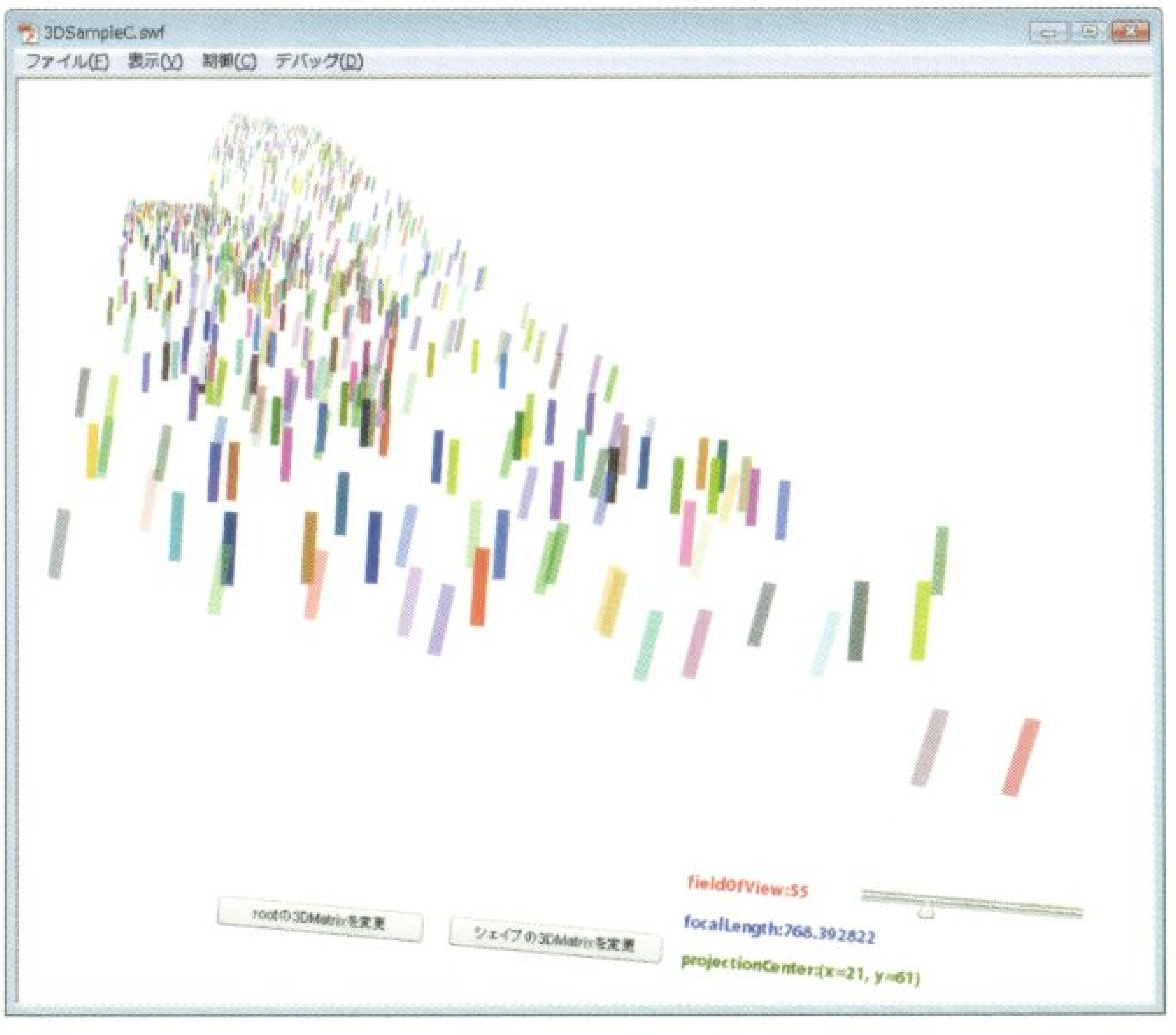
```
function setFieldOfView( e:Event ):void {
  root.transform.perspectiveProjection.fieldOfView=fv.value;
  showInfo();
}
function showInfo():void {
  fv_txt.text="fieldOfView:"+fv.value;
  fl_txt.text="focalLength:"+root.transform.perspectiveProjection.
focalLength;
  pc_txt.text="projectionCenter:"+root.transform.
perspectiveProjection.projectionCenter.toString();// as Point;
}
//表示オブジェクトルートの変換を行う関数(イベントハンドラ)
function setMatrix3Droot(event:MouseEvent):void {
  this.z=0;
  //Matrix3Dオブジェクトを取得
  mat3=root.transform.matrix3D;
  mat3.appendRotation(3, Vector3D.Y_AXIS );
  mat3.appendRotation(3, Vector3D.Z_AXIS );                    ③
  mat3.appendRotation(3, Vector3D.X_AXIS );
  //Matrix3Dオブジェクトを適用
  this.transform.matrix3D=mat3;
}
//シェイプオブジェクトルートの変換を行う関数(イベントハンドラ)
function setMatrix3D(event:MouseEvent):void {
  for (var i:int = 0; i < 1000; i+=2) {
    curShape=shapes[i] as Shape;
    //Matrix3Dオブジェクトを取得
    mat3=curShape.transform.matrix3D;
    mat3.appendRotation(3, Vector3D.Y_AXIS );
    mat3.appendRotation(3, Vector3D.Z_AXIS );                  ④
    mat3.appendRotation(3, Vector3D.X_AXIS );
    //Matrix3Dオブジェクトを適用
    curShape..transform.matrix3D=mat3;
    //区別するためにアルファ値を変更
    curShape.alpha=0.5;
  }
}
init();
```

7-10-2のサンプルを修正してMatrix3Dオブジェクトによる変換を適用するようにしています。「表示オブジェクトのMatrix3Dオブジェクトを作成する方法」にあげた方法のうち、2と3の処理を行うサンプルです。まず2.の方法を行うために、作成したシェイプを格納しておく配列を用意し(①のshapes変数)、作成したシェイプを②で格納しています。④の関数ですべてのシェイプオブジェクトのうち半分にMatrix3Dによる変換を適用しています。表示オブジェクトのルート自体を変換する処理は③の関数で行っているように、まずthis(この場合rootと同じ)オブジェクトのzプロパティに0を代入してMatrix3Dオブジェクトを生成してから、変換のメソッドを適用します。結果のイメージでわかるように、ルートオブジェクトの座標回転を行うと、ステージ全体に変換が適用される形になります。

■ シェイプの3DMatrixを変更



■ rootの3DMatrixを変更



PART 8

FLASH ACTIONSCRIPT HANDBOOK

グラフィックスとフィルター

8-1-1 ビットマップをロードして表示する

Flashではベクターグラフィックと、ビットマップグラフィックのどちらも表示、操作が可能です。ビットマップグラフィックはその目的と圧縮形式などの違いによりいくつかのファイル形式があります。Flashでロード可能なビットマップグラフィック形式を下表に挙げます。ActionScriptでビットマップをサーバからロードして表示するには、Loaderクラスオブジェクトでロードを行い、Bitmapクラスオブジェクトで表示を行います。ロードが終了した場合、LoaderオブジェクトのcontentLoaderInfoプロパティで参照できるLoaderInfoオブジェクトにEvent.COMPLETEイベント（エラーの場合、IOErrorEvent.IO_ERRORイベント）が送出されるので、これらのイベントハンドラでロードしたファイルを処理できます。ロード終了後、Loaderオブジェクトのcontentプロパティでロードした画像データを取得できるので、Bitmapオブジェクトを取り出し、表示処理を行います。

Flashでロード/操作可能なグラフィック形式

形式	概要
BMP（ビットマップ）	Windowsの標準イメージ形式で、通常圧縮を行わないので劣化を防ぐことが可能です。ピクセル形式のイメージは一般に「ビットマップ」と呼ばれますが、BMP形式を意味するものではありません。
JPEG（ジェーペグ）	圧縮を行い、ファイルサイズを小さくできるので、Web上のイメージ、特に写真の表示にもっともよく使用されます。フルカラーと呼ばれる1600万色に対応しています。保存するたびに劣化が生じます。Flash Player 7以前ではプログレッシブJEPG画像を表示することはできません。
GIF（ジフ）	256色以下の色数に限定されるため、ファイルサイズが非常に小さくなります。アイコンやボタン画像などにもっともよく使用されます。
PNG（ピング）	フルカラーに対応し、劣化のない圧縮方式を使用する新しい形式。古いブラウザでは表示できないケースがある。256レベルのアルファ透明度もサポートされます。
TIFF（ティフ）	多くの画像編集アプリが対応している汎用的な形式ですが、自由度が高いため異なる形式が多くの種類が存在します。ブラウザはTIFFの表示はサポートしません。劣化なしの圧縮をサポートします。

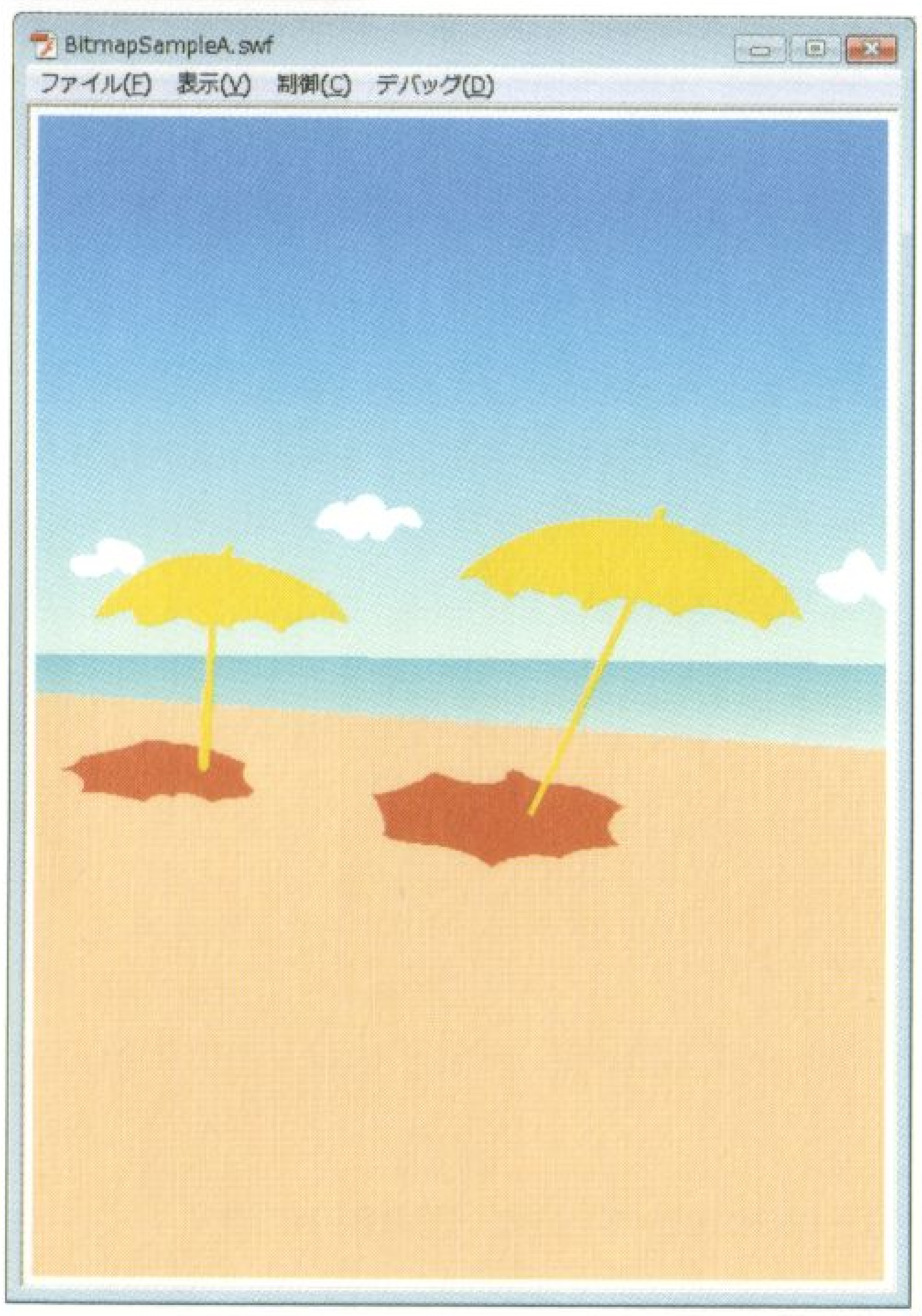
Loaderクラスの主なプロパティ概要

目的	プロパティ名	プロパティの概要
ロードしたファイルを参照する	`content :DisplayObject`	load()メソッドでロードされたファイルのルート表示オブジェクトを参照します。
読み込み状況を提供するLoaderInfoオブジェクトを参照する	`contentLoaderInfo :LoaderInflo`	読み込まれているオブジェクトに対応するLoaderInfoオブジェクトを返します。

Loaderクラスの主なメソッド概要

目的	メソッド名	メソッドの概要
進行中のロード処理をキャンセルする	close() :void	現在進行中のload()メソッドの処理をキャンセルします。
ファイルを指定してロードする	load (request:URLRequest, context:LoaderContext = null):void () :Rectangle	requestで指定したファイルをロードします。
ByteArrayを使用してデータをロードする	loadBytes() :void	ByteArrayオブジェクトに保管されているバイナリデータからロードします。
ロードされたオブジェクトを削除する	unload() :void	load()メソッドを使用して読み込まれたオブジェクトを削除します。
ロードされたSWFを削除してコマンド実行を中止する	unloadAndStop(gc:Boolean=true) :void	ロードされたSWFファイルを削除して、コマンドの実行を中止します。

SAMPLE



SAMPLE CODE

ビットマップをロードしてステージに表示する　BitmapSampleA.fla

```
import myPackage.BitmapSampleClassA;
//ロードするイメージファイル
```

```
var url:String="Images/summerImage1.GIF";  ①
//クラスインスタンスの生成
var obj:BitmapSampleClassA = new BitmapSampleClassA(url);  ②
//表示リストに追加
addChild( obj );  ③
```

このサンプルでは、ビットマップファイル名を指定してBitmapSampleClassAクラスのインスタンスを生成します。①で指定したイメージファイルを②でBitmapSampleClassAクラスのコンストラクタに渡してロードします。その後、③でメインタイムラインの表示リストに追加します。

BitmapSampleClassA.as

```
package myPackage {
  import flash.display.Bitmap;
  import flash.display.Loader;
  import flash.display.Sprite;
  import flash.events.*;
  import flash.net.URLRequest;

  public class BitmapSampleClassA extends Sprite {
    //ロードするイメージファイル
    private var url:String;
    //コンストラクタ
    public function BitmapSampleClassA(url:String) {    ④
      this.url=url;
      doLoad();
    }

    //画像をロードするメソッド
    private function doLoad():void {
      var loader:Loader = new Loader();                   ⑤
      loader.contentLoaderInfo.addEventListener(Event.COMPLETE,
      completeHandler);
      loader.contentLoaderInfo.addEventListener(IOErrorEvent.IO_
      ERROR, ioErrorHandler);
      var request:URLRequest=new URLRequest(url);
      //イメージのロードを実行
      loader.load(request);
```

```
    }

      //ロード完了時のハンドラメソッド
    private function completeHandler(event:Event):void { ——⑥
      var loader:Loader=Loader(event.target.loader);
      //Bitmapオブジェクトとして処理する
      var image:Bitmap=loader.content as Bitmap; ——⑦
      //表示する相対座標を設定
      image.x=5;
      image.y=5; ——⑧ ——⑨
      this.addChild( image );
    }
    private function ioErrorHandler(event:IOErrorEvent):void {
      trace("イメージのロードに失敗しました: " + url);
    }
  }
}
```


このサンプルクラスは、ビットマップイメージをLoaderクラスを使用してロードします。ロードに成功したら、座標を設定して表示リストに追加します。ロードするファイルは④のコンストラクタのパラメータで受け取ります。コンストラクタはurlプロパティにファイル名を設定したら、ロードを行うdoLoad()メソッドを呼び出してロードを開始します。doLoad()メソッドの⑤で、Loaderオブジェクトに対して、ロードが成功したときと失敗したときのハンドラを設定しています。成功した場合、⑥からのハンドラが呼び出されます。その中でビットマップへの参照をimage変数（⑦）に代入し、⑧で座標を設定して、⑨で表示リストに追加します。

8-1-2 ビットマップのロード状況を調べる

ActionScriptでビットマップをサーバからロードして表示する際に、ロードの進行状況を取得できます。進行状況はLoaderInfoオブジェクトで提供され、読み込むファイルの総バイト数、読み込み済みのバイト数やコンテンツのサイズなどが取得できます。LoaderInfoオブジェクトを取得するには、ロードを実行するLoaderオブジェクトのcontentLoaderInfoプロパティを使用します。LoaderInfoオブジェクトは、メインSWFがロードされるときにも生成されます。メインSWFのLoaderInfoオブジェクトを取得するには、表示オブジェクトクラスのloaderInfoプロパティを使用します。ロードの進行状況は、"progress"イベントのイベントオブジェクトである、ProgressEventオブジェクトのプロパティで取得できます。

ProgressEventクラスの主なプロパティ概要

目的	プロパティ名	プロパティの概要
ロード済みサイズを取得する	bytesLoaded : uint	ロードするファイルの読み込み済みバイト数を表します。
ロードするファイルのサイズを取得する	bytesTotal : uint	ロードするファイル全体の圧縮後のバイト数を表します。

LoaderInfoクラスの主なイベント概要

タイミング	イベント名	イベントの概要
ロードが正常に終了した	complete	データが正常にロードされたときに送出されます。
HTTPステータスコードを検出	httpStatus	ロード要求がHTTPで行われ、HTTPステータスコードを検出できる場合に送出されます。
ロードされたSWFのプロパティとメソッドが検出可能になった	init	ロードされたSWFファイルのプロパティおよびメソッドがアクセス可能で使用できる状態になると送出されます。
ロードが失敗した	ioError	ロード処理が失敗したときに送出されます。
ロードを開始した	open	ロードが開始されたときに送出されます。
ロードの進行状況を受信した	progress	ロード処理中に状況を示すデータを受信したときに送出されます。

LoaderInfoクラスの主なプロパティ概要

目的	プロパティ名	プロパティの概要
ロード済みサイズを取得する	bytesLoaded : uint	ロードするファイルの読み込み済みバイト数を表します。ProgressEventクラスにも同様のプロパティがあります。

LoaderInfoクラスの主なプロパティ概要（続き）

目的	プロパティ名	プロパティの概要
ロードするファイルのサイズを取得する	bytesTotal : uint	ロードするファイル全体の圧縮後のバイト数を表します。ProgressEventクラスにも同様のプロパティがあります。
読み込まれたオブジェクトを参照する	content : DisplayObject	LoaderInfoオブジェクトに関係した読み込まれたオブジェクトを参照します。
読み込まれたファイルの幅と高さを取得する	height : int width : int	読み込まれたファイルの規格高さ/規格幅を表します。
読み込んだSWFのフレームレートを取得する	frameRate : Number	読み込み済みのSWFファイルに関するフレームレートを取得できます。
読み込んだSWFのファイル形式バージョンを取得する	swfVersion : uint	読み込み済みのSWFファイルのファイル形式のバージョンを取得できます。

SAMPLE



SAMPLE CODE

ビットマップのロード開始/進行状況を調べる

BitmapSampleB.fla

```
import myPackage.BitmapSampleClassB;
//ロードするイメージファイル
var url:String="Images/summerImage1.GIF";
```

```
//クラスインスタンスの生成
var obj:BitmapSampleClassB = new BitmapSampleClassB(url);
//表示リストに追加
addChild( obj );
```


8-1-1 のBitmapSampleA.flaとほぼ同じです。読み込むイメージファイルを、サイズがやや大きいものにしてprogressイベントを多く発生させています。

BitmapSampleClassB.as

```
package myPackage {
  import flash.display.Bitmap;
  import flash.display.Loader;
  import flash.display.Sprite;
  import flash.events.*;
  import flash.net.URLRequest;
  import flash.events.HTTPStatusEvent;
  import flash.events.ProgressEvent;

  public class BitmapSampleClassB extends Sprite {
    //ロードするイメージファイル
    private var url:String;
    //Loaderオブジェクト変数
    var loader:Loader;

    //コンストラクタ
    public function BitmapSampleClassB(url:String) {
      this.url=url;
      doLoad();
    }

    //画像をロードするメソッド
    private function doLoad():void {
      loader = new Loader();
      loader.contentLoaderInfo.addEventListener(Event.COMPLETE,
      completeHandler);
      loader.contentLoaderInfo.                                  ①
      addEventListener(IOErrorEvent.IO_ERROR, ioErrorHandler);
      loader.contentLoaderInfo.
```

```
    addEventListener(HTTPStatusEvent.HTTP_STATUS,
    httpStatusHandler);
    loader.contentLoaderInfo.addEventListener(Event.INIT, init
    Handler);                                                      ─①
    loader.contentLoaderInfo.addEventListener(Event.OPEN, open
    Handler);
    loader.contentLoaderInfo.addEventListener(ProgressEvent.
    PROGRESS, progressHandler);
    var request:URLRequest=new URLRequest(url);
    //イメージのロードを実行
    loader.load(request);
  }

  //以降はハンドラメソッドの定義
  private function httpStatusHandler(event:HTTPStatusEvent) { ──②
    trace("httpStatusを受信しました: HTTPステータスコード: " + event.status
    );
  }
  private function initHandler(event:Event) {
    trace("initイベントを受信しました" );
  }
  private function openHandler(event:Event) {
    trace("openイベントを受信しました" );
  }
  private function progressHandler(event:ProgressEvent) {
    //ロードの進行状況を調べるには、progressイベントハンドラで処理を行います。
    trace("progressイベントを受信しました: 受信済みバイト数: " + event.
    bytesLoaded + " / 総バイト数: " + event.bytesTotal  +
        " >>LoaderInfoの情報:" + loader.contentLoaderInfo.bytes      ─③
        Loaded + "バイト/" + loader.contentLoaderInfo.bytesTotal
        + "バイト");
  }
  private function completeHandler(event:Event):void {
    trace("completeイベントを受信しました: ファイルのサイズ(height=" + loader.
    height + ", width=" + loader.width + ")" );
    //Bitmapオブジェクトとして処理する
    var image:Bitmap=loader.content as Bitmap;
    //表示する相対座標を設定
    image.x=5;
```

```
            image.y=5;
            this.addChild( image );
        }
        private function ioErrorHandler(event:IOErrorEvent):void {
            trace("イメージのロードに失敗しました: " + url);
        }
    }
}
```


8-1-1 のBitmapSampleClassB.asにイベント処理を追加したものです。イベントハンドラの登録は、①で行っています。ハンドラは②以降ですが、completeHandler()でメッセージの主力と、ファイルの縦横サイズの取得も行っています。ioErrorHandler()は同じものです。③のprogressHandler()がロードの進行状況を示すものになります。trace()関数でprogress状況を出力しています。通常はProgressEventイベントオブジェクトの情報を使用しますが、バイト数を示すLoaderInfoオブジェクトの同じ情報も出力しています。ローカルで実行した結果は下記のようになります。

▶ **実行結果 出力**

```
openイベントを受信しました
progressイベントを受信しました: 受信済みバイト数: 0 / 総バイト数: 200316 >>LoaderInfo
の情報:0バイト/200316バイト
progressイベントを受信しました: 受信済みバイト数: 65536 / 総バイト数: 200316
>>LoaderInfoの情報:65536バイト/200316バイト
progressイベントを受信しました: 受信済みバイト数: 131072 / 総バイト数: 200316
>>LoaderInfoの情報:131072バイト/200316バイト
progressイベントを受信しました: 受信済みバイト数: 196608 / 総バイト数: 200316
>>LoaderInfoの情報:196608バイト/200316バイト
progressイベントを受信しました: 受信済みバイト数: 200316 / 総バイト数: 200316
>>LoaderInfoの情報:200316バイト/200316バイト
initイベントを受信しました
httpStatusを受信しました: HTTPステータスコード: 0
completeイベントを受信しました: ファイルのサイズ(height=640, width=480)
```

8-1-3 ライブラリのイメージを表示する

Bitmapクラスは画像イメージを表示オブジェクトとして処理するためのクラスです。そのため、イメージデータの処理に関してはBitmapクラスには下表に示されるように最小限のプロパティしかありません。ビットマップのピクセル処理、フィルターの適用や描画を行ったり、ライブラリに格納されたビットマップをステージ上にロードしたりする際には、BitmapDataクラスを使用する必要があります。本項目では、ライブラリに格納されたビットマップイメージをBitmapDataオブジェクトによりステージ上に表示するサンプルを解説します。このとき、BitmapDataオブジェクトでイメージのデータを作成し、表示するためにBitmapオブジェクトでラップする必要があります。

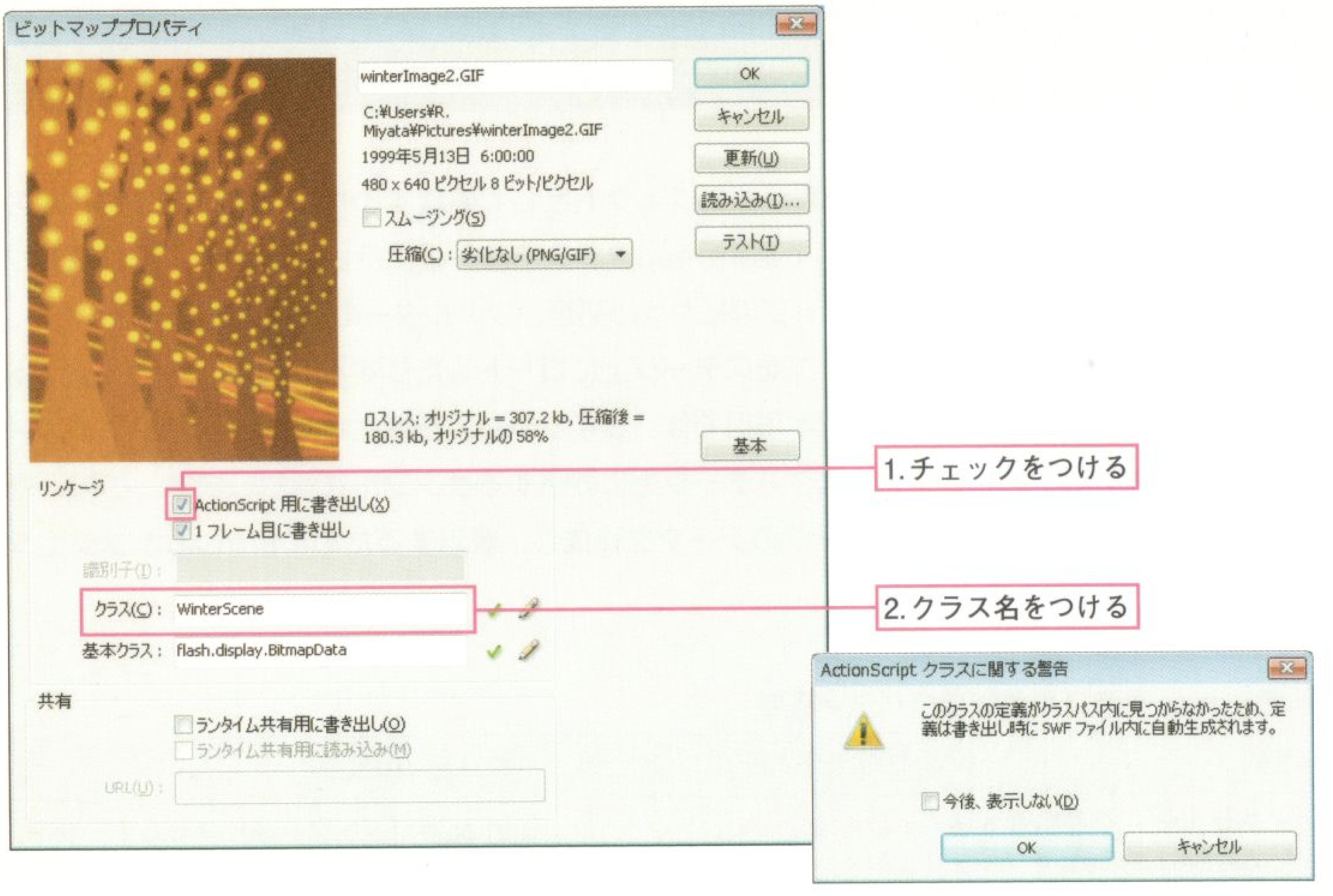
Bitmapクラスの主なプロパティ概要

目的	プロパティ名	プロパティの概要
ピクセルデータを処理するためのBitmapDataオブジェクトを参照する	`bitmapData :flash.display. BitmapData`	Bitmapオブジェクトのピクセルデータを処理するためのBitmapDataオブジェクトにアクセスできます。
もっとも近いピクセルに吸着させる	`pixelSnapping :String`	Bitmapオブジェクトがもっとも近いピクセルに吸着されるかどうかを指定します。
拡大縮小時にスムージングさせる	`smoothing :Boolean`	ビットマップを拡大/縮小するときにスムージングするかどうかを指定します。

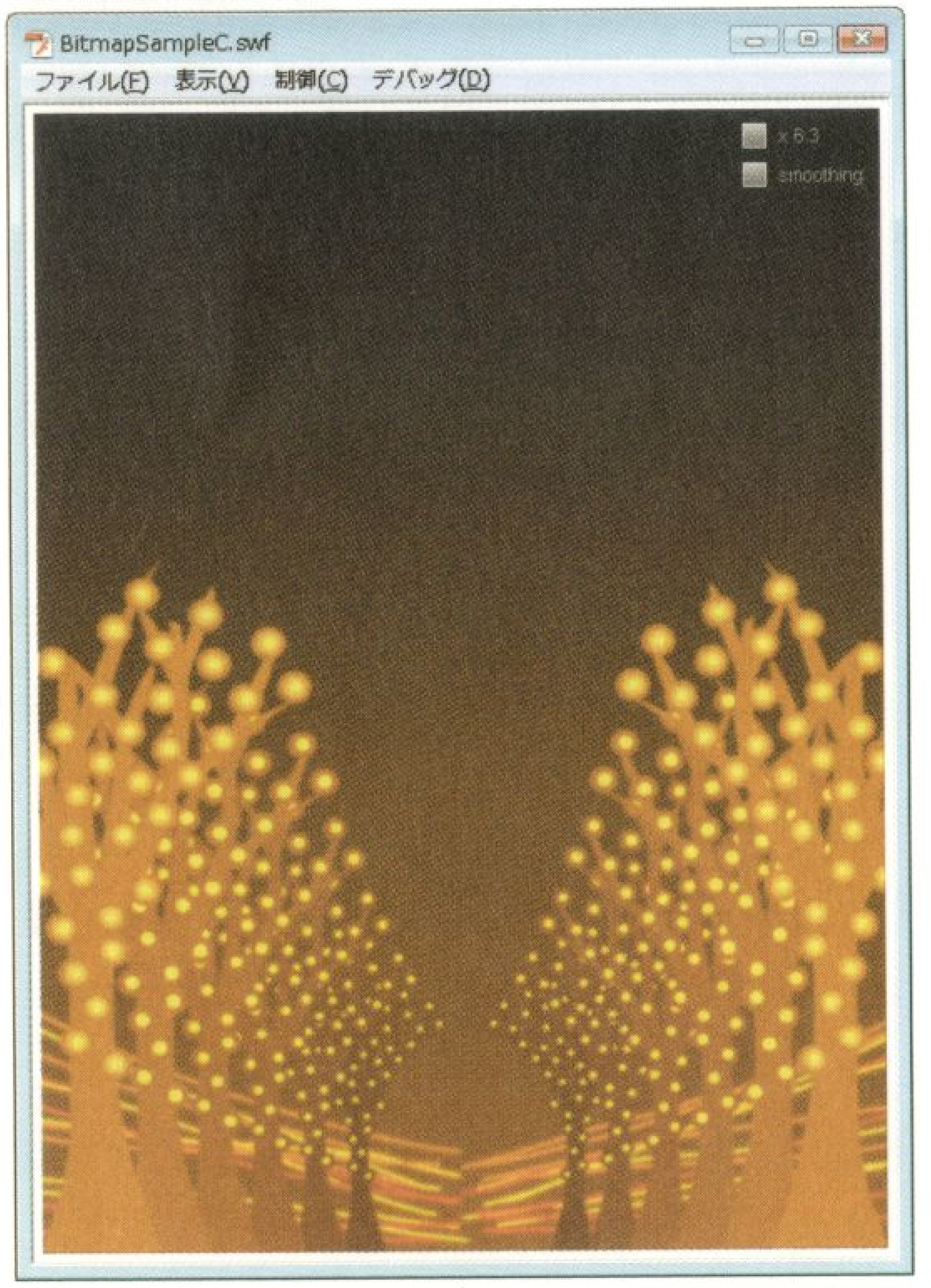
Bitmapクラスの主なメソッド概要

目的	メソッド名	メソッドの概要
Bitmapオブジェクトを作成する	`Bitmap( bitmapData: = null, pixelSnapping:String = "auto", smoothingBoolean = false )`	コンストラクタです。Bitmapオブジェクトを新しく作成します。bitmapDataオブジェクトがすでにある場合、第1パラメータで指定できます。

コードを実行する間にイメージをライブラリに格納してクラス名を設定する必要があります。下図のようにライブラリでイメージを選択してプロパティダイアログボックスを表示して1.「ActionScript用に書き出し」にチェックをつけ、2.「クラス」名を記入します。ここでは、クラス名は「WinterScene」という名称にしています。OKを押してダイアログを閉じる際に「ActionScriptクラスに関する警告」が表示されますが、そのままOKを押してダイアログを閉じます。


ビットマッププロパティ
winterImage2.GIF
C:¥Users¥R.
Miyata¥Pictures¥winterImage2.GIF
1999年5月13日 6:00:00
480 x 640 ピクセル 8 ビット/ピクセル
スムージング(S)
圧縮(C)： 劣化なし (PNG/GIF)
ロスレス: オリジナル = 307.2 kb, 圧縮後 = 180.3 kb, オリジナルの 58%
OK
キャンセル
更新(U)
読み込み(I)...
テスト(T)
基本
リンケージ
ActionScript 用に書き出し(X)
1 フレーム目に書き出し
識別子(I)：
クラス(C)： WinterScene
基本クラス： flash.display.BitmapData
共有
ランタイム共有用に書き出し(O)
ランタイム共有用に読み込み(M)
URL(U)：
1. チェックをつける
2. クラス名をつける
ActionScript クラスに関する警告
このクラスの定義がクラスパス内に見つからなかったため、定義は書き出し時に SWF ファイル内に自動生成されます。
今後、表示しない(D)
OK
キャンセル

SAMPLE


BitmapSampleC.swf
ファイル(F) 表示(V) 制御(C) デバッグ(D)
x 6.3
smoothing

SAMPLE CODE

ライブラリに格納したビットマップイメージをステージに表示する

BitmapSmpleC.fla

```
//ライブラリのイメージインスタンスを格納するBitmapDataオブジェクト変数
var winterSceneData:BitmapData;
//BitmapDataを表示するためのBitmapオブジェクト変数
var winterSceneDisplay:Bitmap;
//ドラッグドロップを行うためのSpriteオブジェクト変数
var container:Sprite;
//初期処理を行う関数
function init():void {
  //ビットマップを表示
  showImage();
  //ドラッグドロップのイベントリスナーを登録
  this.stage.addEventListener(MouseEvent.MOUSE_DOWN, startDragging );
  this.stage.addEventListener(MouseEvent.MOUSE_UP, stopDragging );
  //スムージングをON/OFFにするチェックボックスのイベントリスナー
  smoothing_chk.addEventListener(Event.CHANGE, setSmoothing );
  //ビットマップを拡大するチェックボックスのイベントリスナー
  scaleUp_chk.addEventListener(Event.CHANGE, scaleImage );
}
//ライブラリのイメージを表示する関数
function showImage():void { ―――――――――――――――――――― ①
  //winterSceneというクラス名のビットマップをインスタンス化
  winterSceneData=new winterScene(0,0); ――――――――――― ②
  //BitmapDataインスタンスからBitmapオブジェクトを作成
  winterSceneDisplay=new Bitmap(WinterSceneData);
  //コンテナのSpriteオブジェクト
  container = new Sprite();
  //Bitmapオブジェクトをコンテナに格納
  container.addChild( winterSceneDisplay );
  container.x=5;
  container.y=5;
  //コンテナをステージに表示
  this.addChildAt( container, 0  ); ―――――――――――――― ③
}
//BitmapオブジェクトのsmoothingをON/OFFにする関数
function setSmoothing(event:Event):void {
  winterSceneDisplay.smoothing=event.target.selected;
}
```

```
//イメージのスケールを拡大/縮小する関数
function scaleImage(event:Event):void {
  if (event.target.selected) {
    winterSceneDisplay.scaleX=6.3;
    winterSceneDisplay.scaleY=6.3;
  } else {
    winterSceneDisplay.scaleX=1.0;
    winterSceneDisplay.scaleY=1.0;
  }
}
//コンテナをドラッグする関数
function startDragging(event:MouseEvent):void {
  //チェックボックス上でマウスダウンされた場合、ドラッグしない
  if (event.target==this.scaleUp_chk||event.target==this.
  smoothing_chk) {
    return;
  }
  container.startDrag();
}
//ドラッグを停止する関数
function stopDragging(event:MouseEvent):void {
  //チェックボックス上でマウスアップされた場合、何もしない
  if (event.target==this.scaleUp_chk||event.target==this.
  smoothing_chk) {
    return;
  }
  container.stopDrag();
}
//初期処理関数の呼び出し
init();
```

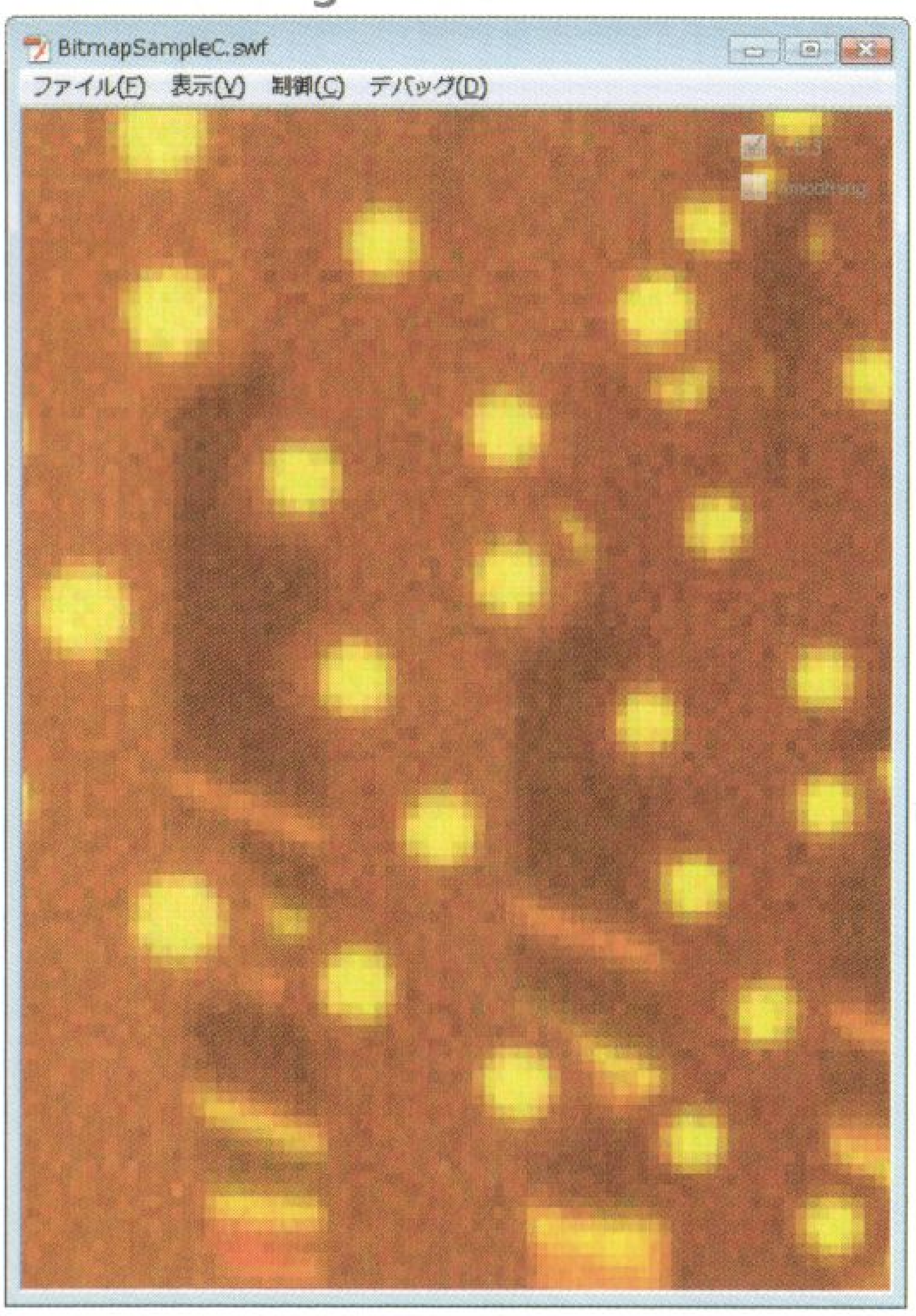
このサンプルでは、ライブラリのビットマップイメージをステージに表示しますが、Bitmapクラスのsmoothingプロパティの効果も確認します。そのためにステージ右上にビットマップを拡大するためのチェックボックスと、smoothingをオンにするチェックボックスを配置しています。拡大したビットマップの見える位置を変えてsmoothing効果を確認できるように、ドラッグできるようにしています。そのため、containerというSpriteオブジェクトを作成して、Bitmapオブジェクトを格納しています。①からのshowImage()関数がライブラリの

ビットマップをステージの表示リストに追加する処理を行っています。②でBitmapDataインスタンスを作成していますが、引数の(0,0)はBitmapDataコンストラクタのパラメータで、本来はビットマップサイズを意味しますが、ライブラリ読み込みの場合は無視されるので、0にしています。③でthis.addChildAt(container, 0) としているのは、ビットマップをチェックボックスの背面に表示するためです。実際には、ドラッグ処理のためのSpriteを除いて記述できます。もっともシンプルな記述は次のようになります。

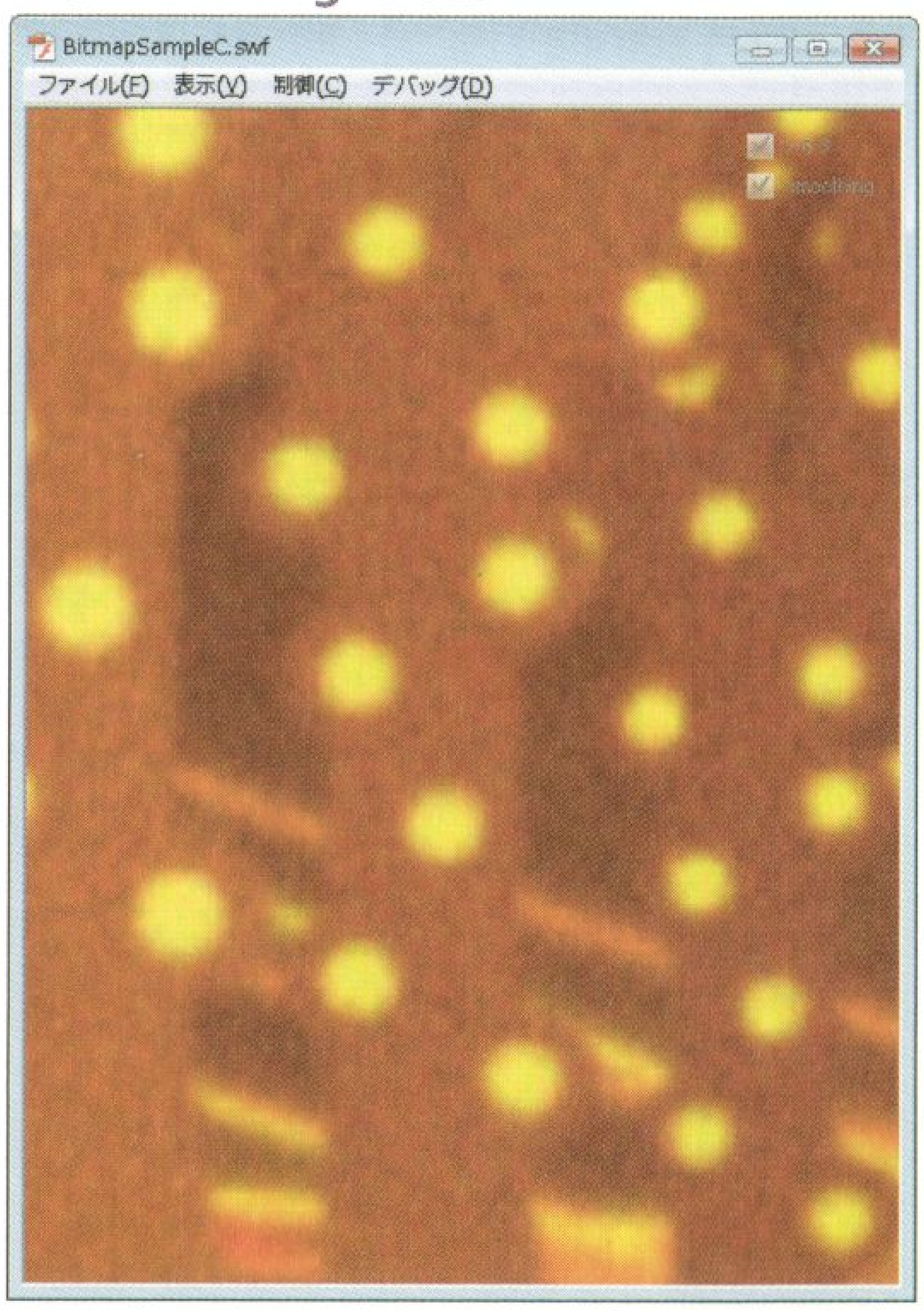
```
//winterSceneというクラス名のビットマップをインスタンス化
winterSceneData=new WinterScene(0,0);
//BitmapDataインスタンスからBitmapオブジェクトを作成
winterSceneDisplay=new Bitmap(winterSceneData);
//Biamtpをステージに表示
this.addChild(winterSceneDisplay  );
```

チェックボックスをクリックすると、イメージが拡大されますが、smoothing=falseの場合とtrueの違いは下記のサンプルのようにギザギザがスムーズ化されることです。

■ チェックボックスをクリックして拡大（smoothing=false）



■ チェックボックスをクリックして拡大（smoothing=true）



8-1-4 BitmapDataを使う

BitmapDataオブジェクトは画像イメージのデータそのものを表し、ピクセルレベルの処理を行うためのメソッドが用意されています。BitmapDataオブジェクトは個別にインスタンスを生成してから、レンダリング用にBitmapオブジェクトに割り当てることもできますし、すでにBitmapオブジェクトがある場合、bitmapDataプロパティで参照することができます。BitmapDataクラスのプロパティと、フィルター、塗りや描画、その他のメソッドは主に下表のものがあります。BitmapDataオブジェクトを表示するには、8-1-3で解説したように、Bitmapオブジェクトでラップする必要があります。サンプルコードでピクセルの取り出しと設定を行っています。この処理を行うメソッドの詳細を下記の表に示します。

BitmapDataクラスの主なプロパティ概要

目的	プロパティ名	プロパティの概要
イメージのサイズを調べる	`height :int` `width :int`	ビットマップイメージの高さと幅(ピクセル単位)を表します。
イメージを表す矩形オブジェクトを取得する	`rect :Rectangle`	[読み取り専用]ビットマップイメージのサイズと位置を表す矩形オブジェクトです。
イメージが透明度をサポートするか調べる	`transparent :Boolean`	[読み取り専用]ビットマップイメージがピクセル単位の透明度をサポートするかを表します。

BitmapDataクラスの主なメソッド概要(フィルター関連)

目的	メソッド名	メソッドの概要
フィルターオブジェクトを適用する	`applyFilter( sourceBitmapData:BitmapData, sourceRect:Rectangle, destPoint:Point, filter:BitmapFilter ):void`	ソースイメージにフィルターオブジェクトで定義されたフィルターを適用します。
applyFilter() メソッドによって影響を受ける矩形を調べる	`generateFilterRect( sourceRect:Rectangle, filter:BitmapFilter ):Rectangle`	ソース矩形とフィルタオブジェクトを指定して、applyFilter() メソッド呼び出しによって影響を受ける矩形を調べて返します。

BitmapDataクラスの主なメソッド概要（塗り、描画関連）

目的	メソッド名	メソッドの概要
ビットマップイメージ上に描画する	`draw( source:IBitmapDrawable, matrix:Matrix = null, colorTransform:ColorTransform = null, blendMode:String = null, clipRect:Rectangle = null, smoothing:Boolean = false ): void`	BitmapDataオブジェクト上にsource で与えられる表示オブジェクトを描画します。
矩形領域のピクセルを塗りつぶす	`fillRect( rect:Rectangle, color:uint ): void`	rectで指定された領域をcolorで示されるARGBカラーで塗りつぶします。
塗りつぶし処理を行う	`floodFill( x:int, y: int, color: uint ):void`	(x, y) 座標を始点としてcolorで塗りつぶしを行います。
ランダムノイズで塗りつぶす	`noise( randomSeed:int, low:uint = 0, high:uint = 255, channelOptions:uint = 7, grayScale:Boolean = false ):void`	パラメータにより指定されるランダムノイズを表すピクセルでイメージを塗りつぶします。
Perlinアルゴリズムで自然に見えるランダムノイズを適用する	`perlinNoise( baseX:Number, baseY:Number, numOctaves:uint, randomSeed:int, stitch:Boolean, fractalNoise:Boolean, channelOptions:uint = 7, grayScale:Boolean = false, offsets:Array = null ):void`	Perlinノイズ生成アルゴリズムによるノイズパターンをビットマップに適用します。
ピクセルディゾルブを実行する	`pixelDissolve( sourceBitmapData: BitmapData, sourceRect:Rectangle, destPoint:Point, randomSeed: int = 0, numPixels:int = 0, fillColor:uint = 0 ):int`	ソースイメージからターゲットイメージへのピクセルディゾルブ、または同じイメージを使用したピクセルディゾルブを実行します。
1ピクセルのカラーを設定する	`setPixel( x:int, y:int, color:uint ):void`	BitmapDataオブジェクトの1つのピクセルカラーを設定します。

PART
1
2
3
4
5
6
7
8
9

BitmapDataクラスの主なメソッド概要（塗り、描画関連）（続き）

目的	メソッド名	メソッドの概要
1ピクセルのカラーとアルファを設定する	setPixel32(x:int, y:int, color:uint):void	BitmapData オブジェクトの1つのピクセルにカラー値とアルファ透明度を設定します。

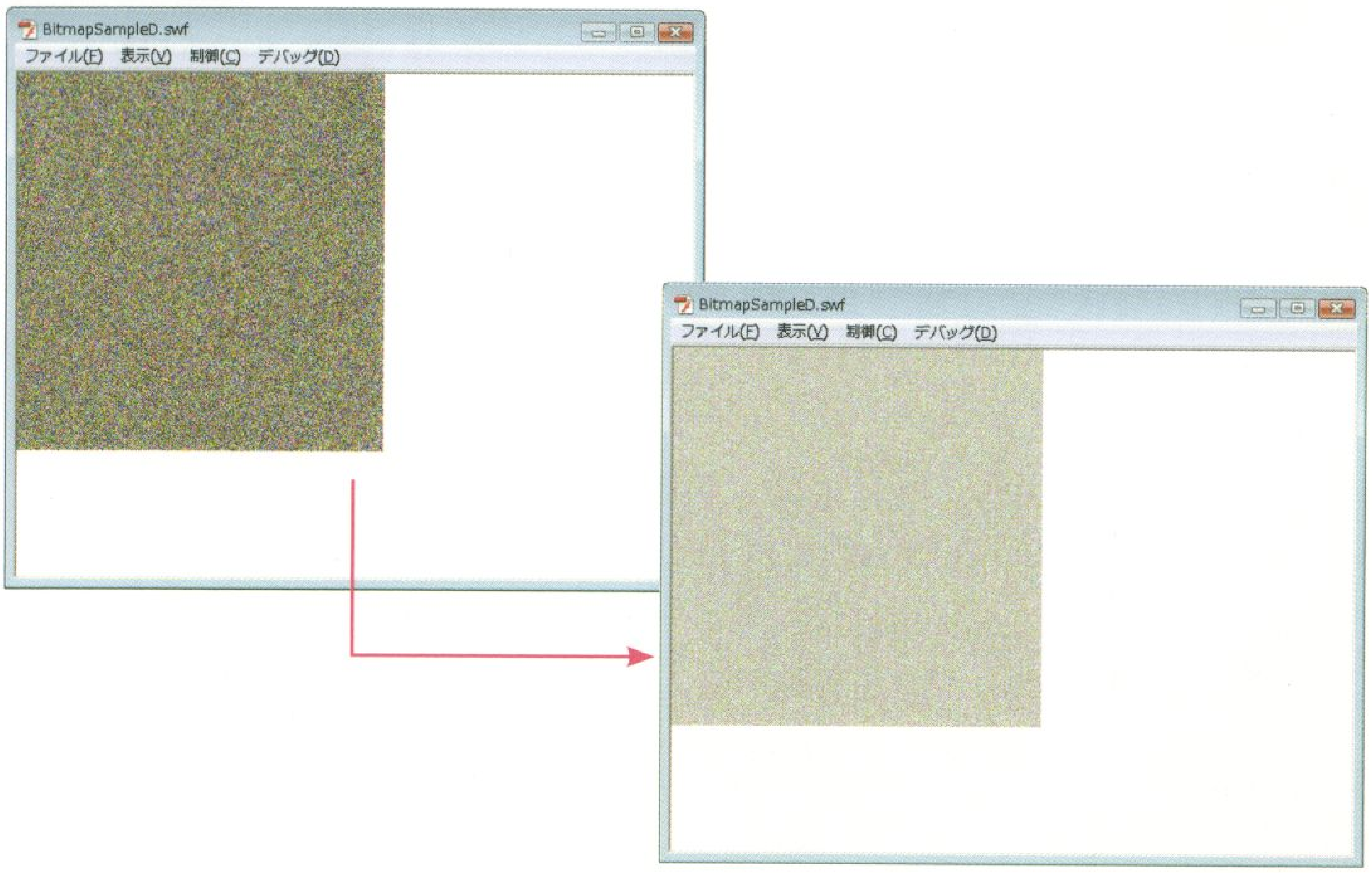
BitmapDataクラスの主なメソッド概要（その他）

目的	メソッド名	メソッドの概要
ビットマップのコピーを作成する	clone():BitmapData	BitmapDataオブジェクトのクローンを作成して新しいオブジェクトとして返します。
ビットマップの一部のコピーを作成する	copyPixels(sourceBitmapData:BitmapData, sourceRect:Rectangle, destPoint:Point, alphaBitmapData: BitmapData = null, alphaPoint:Point = null, mergeAlpha:Boolean = false):void	sourceBitmapDataで与えられるイメージをdestPointで指定される矩形領域にコピーします。
ビットマップのメモリを解放する	dispose():void	BitmapDataオブジェクトに使用されているメモリを解放します。
特定ポイントのRGBピクセル値を調べる	getPixel(x:int, y:int):uint	指定したポイント(x, y)のRGBピクセル値を表す整数を返します。
特定ポイントのARGBピクセル値を調べる	getPixel32(x:int, y:int):uint	指定したポイント(x, y)のアルファチャンネルデータとRGBデータを含むARGBカラー値を返します。
矩形領域からバイト配列を作成する	getPixels(rect:Rectangle):ByteArray	rectで指定される矩形領域からバイト配列を生成します。
バイト配列を矩形領域のピクセルデータに変換する	setPixels(rect:Rectangle, inputByteArray:ByteArray):void	バイト配列のデータで、BitmapDataの矩形領域にあるピクセルを塗りつぶします。
他のビットマップやオブジェクトとの接触を調べる	hitTest(firstPoint:Point, firstAlphaThreshold:uint, secondObject:Object, secondBitmapDataPoint:Point = null, secondAlphaThreshold:uint = 1):Boolean	secondObjectで与えられるポイント、矩形、または他のビットマップイメージとピクセルのヒットがあるかを検出します。
イメージをスクロールする	scroll(x:int, y:int):void	(x, y)で指定したピクセル分イメージをスクロールします。

BitmapDataクラスのピクセル取得/セットを行うメソッド詳細

メソッド名	メソッドの詳細
getPixel(x:int, y:int):uint	x, yで指定したBitmapDataオブジェクト内のポイントのRGBピクセル値を取得して、整数値で返します。ピクセル値はアルファ情報を含まないもので、次のフォーマットです(0xは16進表記を表します)。 0x000000～0xFFFFFF RGBカラー情報の内容は次のとおりです。 0x 00 00 00 R G B
getPixel32(x:int, y:int):uint	x, yで指定したBitmapDataオブジェクト内のポイントのARGBピクセル値を取得して、整数値で返します。ARGBピクセル値はアルファ情報を含むもので、次のフォーマットです(0xは16進表記を表します)。 0x00000000～0xFFFFFFFF ARGBカラー情報の内容は次のとおりです。 0x 00 00 00 00 ｱﾙﾌｧ R G B
setPixel(x:int, y:int, color:uint):void	x, yで指定したBitmapDataオブジェクト内のポイントのピクセル値をcolorで指定したRGB値にします。
setPixel32(x:int, y:int, color):void	x, yで指定したBitmapDataオブジェクト内のポイントのピクセル値をcolorで指定したARGB値にします。

SAMPLE



初期状態では、各ピクセルにランダムな値を持つ300×300の正方形を描画し、タイマー処理により、アルファ値を減少させてフェードします。

SAMPLE CODE

BitmapDataによる図形の作成とアルファ値の変更

BitmapSampleD.fla

```
import flash.display.Bitmap;
import flash.display.BitmapData;
import flash.utils.Timer;

//BitmapDataオブジェクト変数
var bitmapData:BitmapData;
//Bitmapオブジェクト変数
var bitmap:Bitmap;
//描く正方形の縦横サイズ
var hw:int=300;
var i:int,j:int;
//ピクセルデータの格納用変数
var pix:uint;
//アルファ値の格納用変数
var alp:uint;
//タイマーオブジェクト変数
var tmr:Timer;

//初期処理(正方形のBitmapDataを作成して表示する)
function init() {
  //正方形のBitmapDataを作成
  bitmapData=new BitmapData(hw,hw,true);
  for (i = 0; i <= hw; i++) {
    for (j = 0; j <= hw; j++) {
      //ピクセルカラーをランダムな値にする
      bitmapData.setPixel( i, j, Math.random() * 0xFFFFFF );
    }
  }
  //表示用のBitmapオブジェクトをBitmapDataオブジェクトから作成する
  bitmap=new Bitmap(bitmapData);
  //表示リストにBitmapオブジェクトを追加
  this.addChild( bitmap );
  //Timerオブジェクトとイベントリスナーの登録
```

```
  tmr=new Timer(10);
  tmr.addEventListener(TimerEvent.TIMER, setFade);
  tmr.addEventListener(TimerEvent.TIMER_COMPLETE, endFade);
  tmr.start();
}

//Timerイベントで呼び出されるリスナー
function setFade(event:Event):void {
  for (i = 0; i <= hw; i++) {
    for (j = 0; j <= hw; j++) {
      //ピクセルのARGB値を取得
      pix=bitmapData.getPixel32(i,j);
      //アルファ値を取り出す
      alp = ( pix & 0xFF000000 ) >>> 24;
      //アルファ値を-1
      alp -= 1;
      //アルファ値が0になったら終了する
      if (alp==0) {
        trace( "終了しました。!" );
        tmr.dispatchEvent( new TimerEvent( TimerEvent.TIMER_COMPLETE
        ) );
        return;
      }
      //元のピクセル値と新しいアルファ値で新しいピクセル値を作成
      pix = ( alp << 24 ) | ( pix & 0x00FFFFFF );
      //BitmapDataオブジェクトに設定
      bitmapData.setPixel32( i, j,  pix );
    }
  }
}

//Timerイベント処理を終了する関数
function endFade(event:Event):void {
  tmr.removeEventListener(TimerEvent.TIMER, setFade);
}

//初期処理の呼び出し
init();
```

BitmapDataで図形を作成して表示、ピクセル処理を行うサンプルです。①からのinit()関数でBitmapDataオブジェクトの作成とBitmapオブジェクトによる表示リストへの追加を行っています。②で行っている処理は、作成した四角形のピクセルを1つずつランダムな値にしています。実行結果イメージにあるような、1ピクセルずつ色が異なるノイズ風の塗りになります。③の処理は、BitmapDataオブジェクトをメインタイムラインの表示リストに追加してステージに表示するものです。④からの関数はTimerイベントで呼び出されるものですが、やはり1ピクセルごとの処理を行っています。⑤でgetPixel32()メソッドにより、アルファ値を含むピクセルデータを取得しています。⑥の処理は、ARGBの32ビットのうち、アルファ値を表す上位8ビットを取得するものです。⑦で値を-1します。その結果、0になった場合、処理を終了するために⑧の処理を行い、Timerのキャンセルを行うためのTIMER_COMPLETEイベントの送出を行います。⑨で元のピクセル値と、新しいアルファ値を合わせて新しいピクセル値を作成してBitmapDataオブジェクトの指定したピクセル位置に適用します。

タイマー終了時のメッセージ

```
終了しました。!
```

8-1-5 ビットマップイメージをコピーする

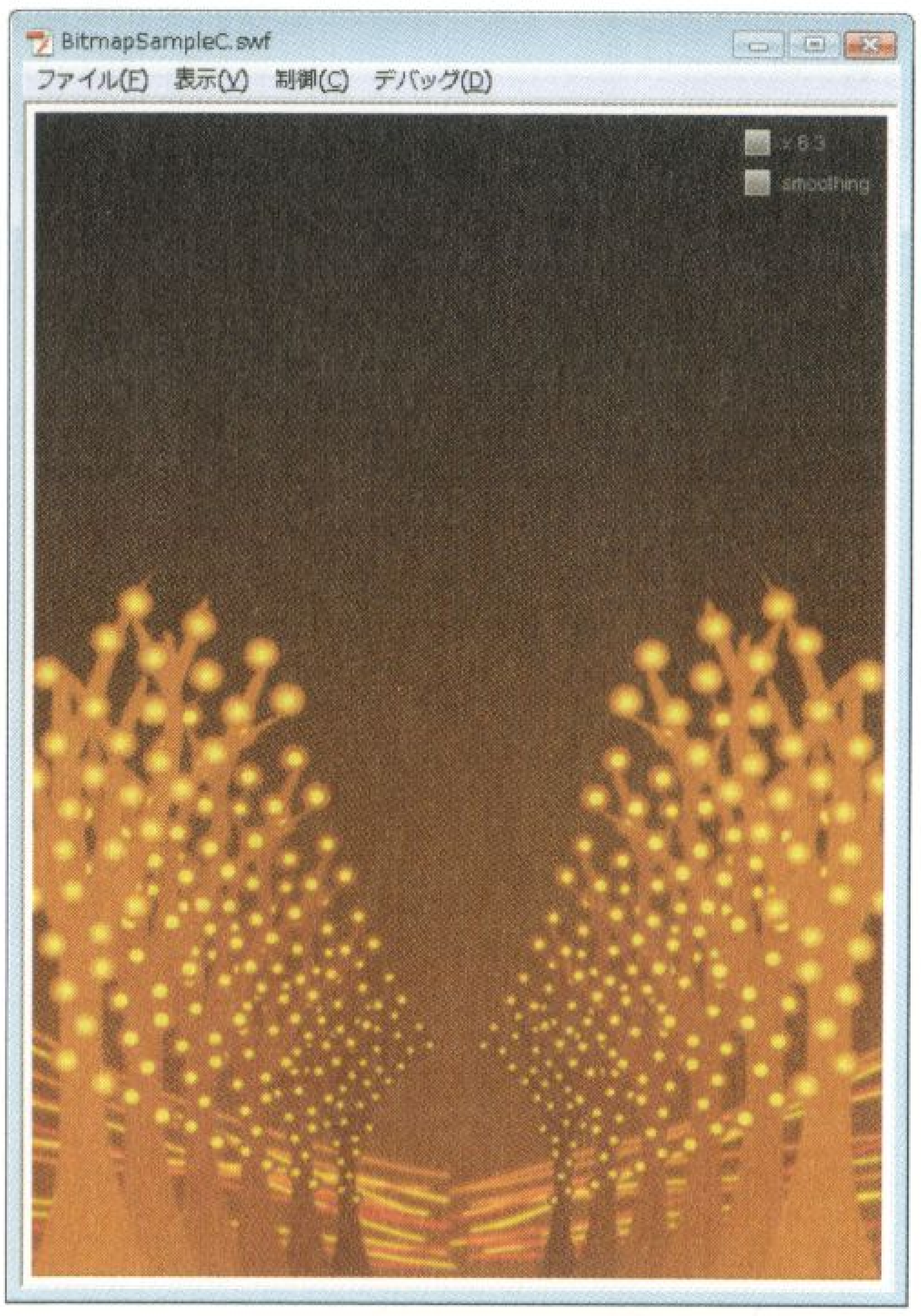
BitmapDataクラスにはビットマップデータを操作するためのメソッドが用意されていますが、ビットマップデータをコピーする場合はclone()メソッドを使用します。このメソッドは文字通りクローンを作成して返すもので、BitmapData型の変数に戻り値を代入してすぐにコピー元と同じビットマップを操作できます。その後ビットマップに描画やフィルターをかける場合、BitmapDataクラスのメソッドを使用できますが、サイズやアルファ、位置などの操作は表示オブジェクトと共通の処理になるため、Bitmapオブジェクトでラップしてから処理を行います。こちらのサンプルでは、コピーしたビットマップの大きさをアルファ、位置をBitmapオブジェクトで変更します。

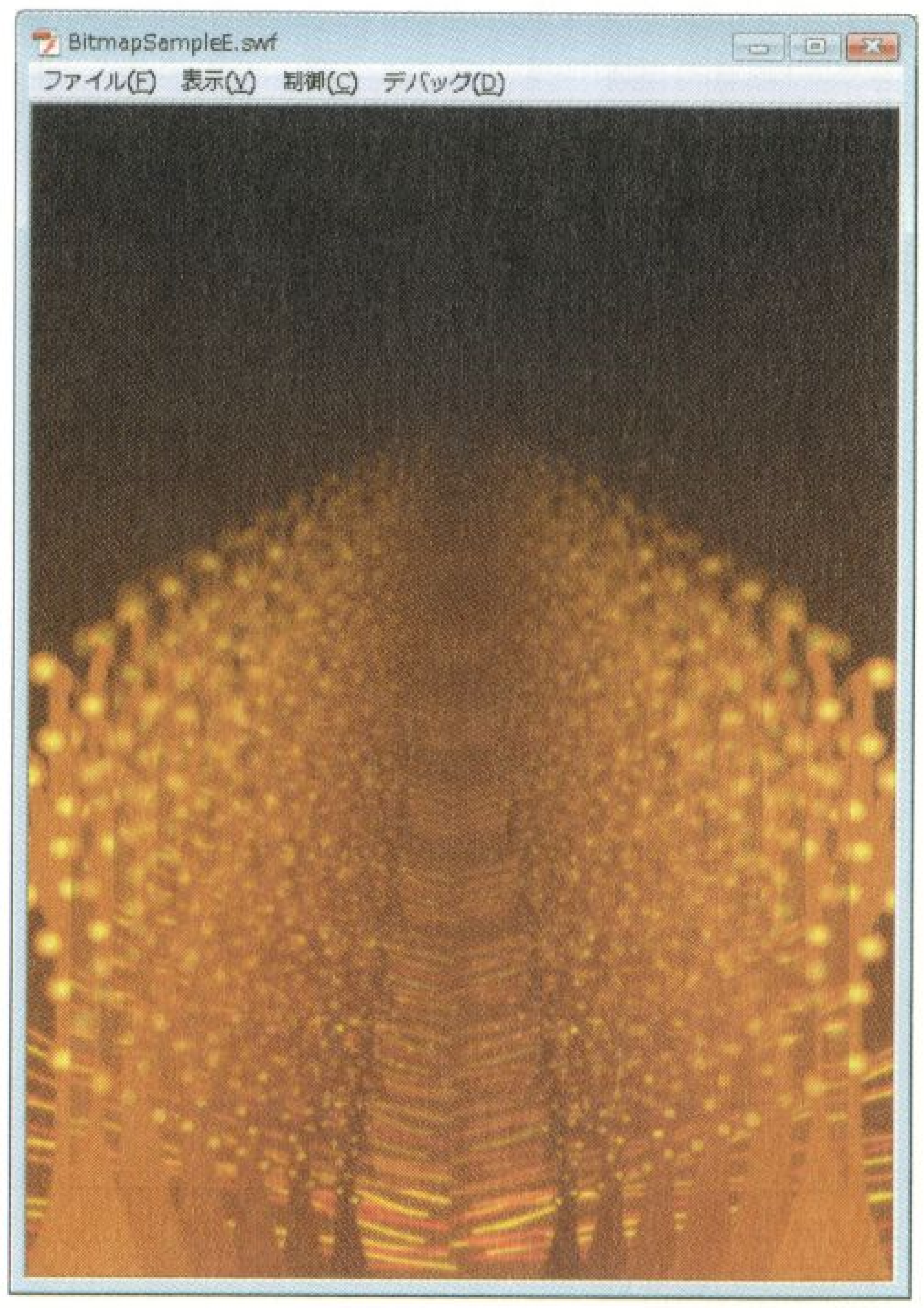
BitmapData クラスのコピーメソッド詳細

メソッド名	メソッドの詳細
clone() :BitmapData	現在のBitmapDataオブジェクトと同じ内容の新しいオブジェクトを作成して返します。現在オブジェクトをセーブしておき、編集する場合などにも使用できます。

SAMPLE

元のイメージ





SAMPLE CODE

BitmapDataオブジェクトのコピーを多数作成する

BitmapSampleE.fla

```
//ライブラリのイメージインスタンスを格納するBitmapDataオブジェクト変数
var winterSceneData:BitmapData;
//BitmapDataを表示するためのBitmapオブジェクト変数
var winterSceneDisplay:Bitmap;
//BitmapDataのクローン格納用変数
var cloneData:BitmapData;
//クローン用表示オブジェクト
var cloneDisp:Bitmap;
//初期処理を行う関数
function init():void {
  //ビットマップを表示
  showImage();
}
//ライブラリのイメージを表示する関数
function showImage():void {
  //winterSceneというクラス名のビットマップをインスタンス化
  winterSceneData=new WinterScene(0,0);
  //BitmapDataインスタンスからBitmapオブジェクトを作成
  winterSceneDisplay=new Bitmap(winterSceneData);
  this.addChildAt( winterSceneDisplay, 0  );
  for ( var i:int = 1 ; i <= 10 ; i++) {
    cloneData = winterSceneData.clone();
    cloneDisp = new Bitmap( cloneData );
    cloneDisp.alpha = Number( ( 10 - i ) / 20 );
    cloneDisp.scaleX = Number( ( 10 - i ) / 10 );
    cloneDisp.scaleY = Number( ( 10 - i ) / 10 );
    cloneDisp.x = ( stage.stageWidth - cloneDisp.width ) / 2;
    cloneDisp.y = ( stage.stageHeight - cloneDisp.height ) / 4;
    cloneDisp.smoothing = true;
    addChild( cloneDisp );
  }
}
//初期処理関数の呼び出し
init();
```

①ライブラリイメージの表示

②ライブラリイメージの表示

③10個のクローンを作成。違いがわかるようにalpha、scaleX、scaleYの位置変更

BitmapSampleC.flaで使用したライブラリ格納のイメージを再び使用して、クローンを多数作成するサンプルです。

8-1-6 表示オブジェクトをビットマップイメージ上に描画する

BitmapDataクラスには、ビットマップイメージ上に、指定した表示オブジェクトを描画するためのdraw()メソッドがあります。このメソッドのパラメータには、描画する表示オブジェクト/ビットマップデータ以外に、描画時に適用するMatrixオブジェクト(ビットマップの変換)、ColorTransformオブジェクト(ビットマップのカラー調整)、ブレンドモード、描画領域、スムージングを指定できます。つまり、描画するオブジェクトのマトリックス操作その他のアレンジが行えます。タイムラインで編集すればただインスタンスを配置してプロパティを設定するだけですが、ActionScriptで動的に描画するためには、サンプルコードのような処理を行います。

BitmapDataクラスのコピーメソッド詳細

メソッド名	メソッドの詳細
`draw( source: IBitmapDrawable, matrix:Matrix = null, colorTransform: ColorTransform = null, blendMode:String = null, clipRect:Rectangle = null, smoothing:Boolean = false ):void`	現在のBitmapDataオブジェクト上にsourceで指定したオブジェクトを描画します。このパラメータの型はIBitmapDrawableとなっていますが、BitmapDataとDisplayObjectクラスが実装するインタフェースで、パラメータにはこのどちらかのオブジェクトを渡すことができます。 **matrix**：描画するsourceオブジェクトの位置を移動、拡大/縮小、回転、変換する場合、それらの設定を行ったMatrixオブジェクトを指定できます。 **colorTransform**：sourceオブジェクトのカラーを調整するためのColorTransformオブジェクトを指定できます。 **blendMode**：flash.display.BlencModeクラスの定数値を指定してブレンドモードを適用できます。 **clipRect**：sourceのうち、描画する領域をRectangleオブジェクトで指定できます。指定しないとsourceオブジェクト全体が描画されます。 **smoothing**：trueの場合、sourceオブジェクトがBitmapDataの場合、サイズ変更や回転時のスムージングを行うことができます。

1
6

SAMPLE



クリックして描画後



SAMPLE CODE

BitmapDataオブジェクト上にビットマップとテキストフィールドを描画する BitmapSampleE.fla

```
import flash.display.BitmapData;
import flash.display.Bitmap;
import flash.text.TextField;
import flash.text.TextFormat;
import flash.geom.Matrix;
import flash.display.BlendMode;

//背景のイメージインスタンスを格納するBitmapDataオブジェクト変数
var winterSceneData:BitmapData;
//BitmapDataを表示するためのBitmapオブジェクト変数
var winterSceneDisplay:Bitmap;
//描画するイメージインスタンスを格納するBitmapDataオブジェクト変数
var flowerData:BitmapData;
//描画するテキストオブジェクト変数
var textData:TextField;
//テキストオブジェクトのフォーマット用オブジェクト変数
var textFormat:TextFormat;
//マトリックスオブジェクト変数
var mtx1:Matrix;
var mtx2:Matrix;

//初期処理を行う関数
function init():void {
  //ビットマップを表示
  showImage();
}

//ライブラリのイメージを表示する関数
function showImage():void {
  //WinterSnowというクラス名のビットマップをインスタンス化
  winterSceneData=new WinterSnow(0,0);
  //BitmapDataインスタンスからBitmapオブジェクトを作成
  winterSceneDisplay = new Bitmap(winterSceneData);
  this.addChildAt( winterSceneDisplay, 0  );
  //マウスをクリックしたときに描画を行います。
  this.stage.addEventListener( MouseEvent.CLICK, clicked );
}
```

```
//マウスのクリックイベントハンドラ
function clicked(event:MouseEvent):void { ─── ①
  //描画するBitmapData
  flowerData=new FlowerImage(0,0); ─── ②
  mtx1 = new Matrix();
  //位置を(280,230)に移動
  mtx1.translate( 280, 230 ); ─── ③
  //draw()メソッドで描画
  winterSceneData.draw( flowerData, mtx1);
  //描画するTextFieldとTextFormat
  textData = new TextField(); ─── ④
  textData.autoSize=TextFieldAutoSize.LEFT;
  textFormat = new TextFormat(); ─── ⑤
  textFormat.font = "Monotype Corsiva";
  textFormat.size = 30;
  textFormat.color = 0xFFFFFF;
  textData.defaultTextFormat = textFormat;
  textData.text = "Merry X'mas";
  mtx2 = new Matrix();
  //TextFieldの位置を(20,14)に移動
  mtx2.translate( 20, 14 );
  //TextFieldを描画
  winterSceneData.draw( textData, mtx2 ); ─── ⑥
}

//初期処理関数の呼び出し
init();
```

今回のテーマはBitmapDataオブジェクトのメソッドにより、動的に描画を行うことなので、サンプルでは、ステージをクリックしたタイミングで描画を行います。①からの関数が描画を行うものです。②で作成しているBitmapDataオブジェクトはライブラリに作成したビットマップイメージクラスです。③のようにMatrixオブジェクトを作成してdraw()メソッドに渡すことで位置やスケール、回転なども適用できます。④で作成しているTextFieldオブジェクトはテキストを描画するためのものです。⑤のようにTextFormatオブジェクトにより、書式を設定してから、⑥で描画を行っています。

8-2 フィルター

8-2-1 フィルターオブジェクトを作成して表示オブジェクトに適用する

flash.filtersパッケージのフィルタークラスオブジェクトを作成して、DisplayObjectまたはBitmapDataオブジェクトにフィルターを適用することができます。DisplayObjectへのフィルターの適用手順はシンプルですが、配列を使用してフィルターオブジェクトを累積的に適用できることと、オブジェクトに直接フィルターが適用されることが特徴的なことです。ActionScript3.0では、下表のようなフィルターが使用可能です。

DisplayObjectへのフィルター適用手順

順番	処理内容
1	適用するすべてのフィルターのインスタンスを作成し、プロパティを設定する。
2	適用先のDisplayObjectのfiltersArrayプロパティの値をArray型変数に代入し、1で作成したフィルターインスタンスを格納する。
3	DisplayObjectオブジェクトのfiltersプロパティに2で作成したArrayオブジェクトを再設定する。適用後、Arrayオブジェクトに格納されたフィルターが累積的に適用されます。

ActionScript3.0のフィルター

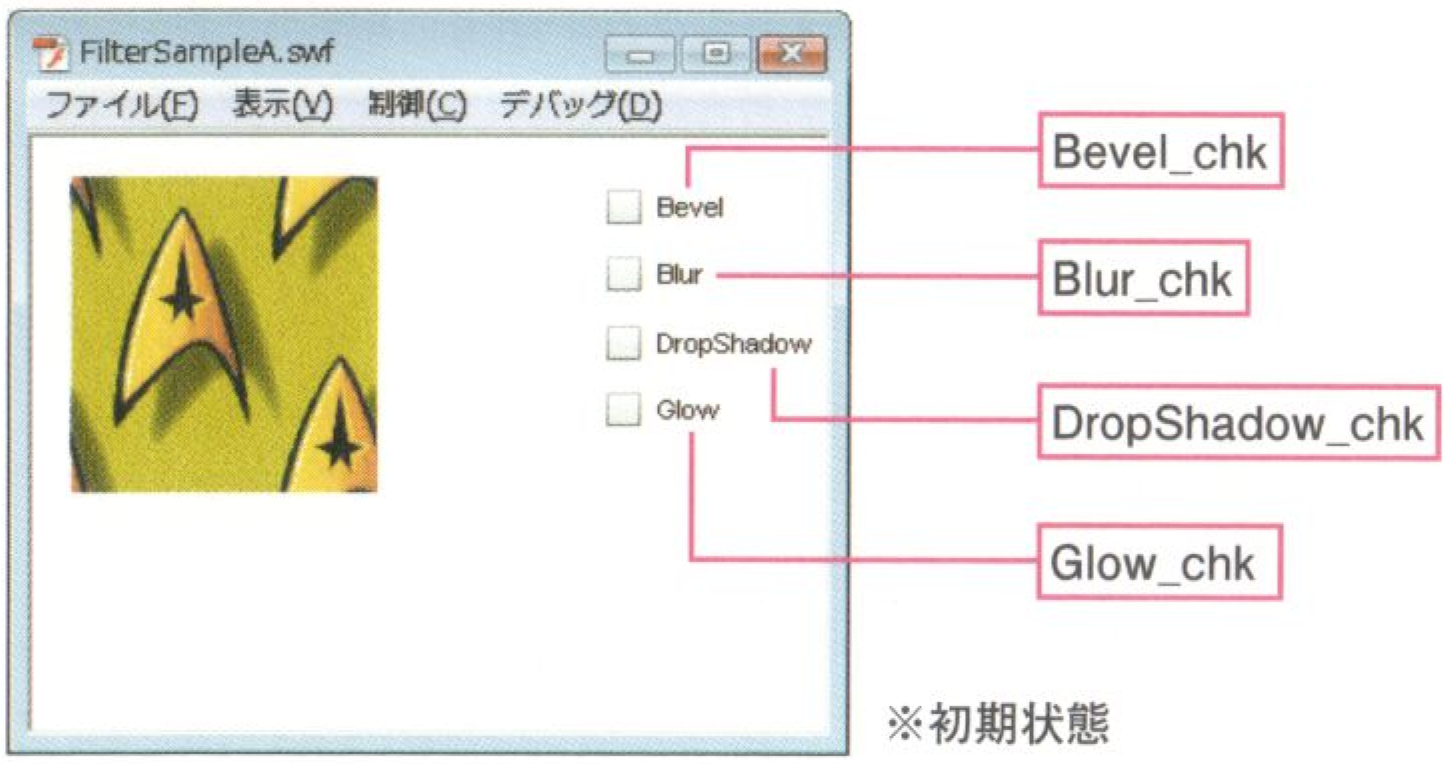
フィルター	クラス	フィルターの内容	Flash
ベベル	BevelFilter	表示オブジェクトにベベル効果を追加できます。	○
ぼかし	BlurFilter	表示オブジェクトにぼかし効果を適用できます。	○
カラーマトリックス	ColorMatrixFilter	入力イメージの各ピクセルのRGBAカラー値とアルファ値に4 × 5マトリックス変換を適用することで、新しいRGBAカラー値とアルファ値からなる結果を作成できます。	
畳み込み	ConvolutionFilter	マトリックス畳み込みフィルター効果を適用できます。	
置き換えマップ	DisplacementMapFilter	指定されたBitmapDataオブジェクト(置き換えマップイメージといいます)のピクセル値を使用して、オブジェクトの置き換え(変位)を実行します。	
ドロップシャドウ	DropShadowFilter	ドロップシャドウを表示オブジェクトに追加します。	○
グロー	GlowFilter	表示オブジェクトにグロー効果を適用できます。	○
グラデーションベベル	GradientBevelFilter	オブジェクトにグラデーションベベル効果を適用し、表示できます。	○
グラデーショングロー	GradientGlowFilter	表示オブジェクトにグラデーショングロー効果を適用できます。	○
シェーダー	ShaderFilter	フィルター処理されるオブジェクトにシェーダを実行することによってフィルターを適用します。	

Flash列の○は、Flash CS4/CS5 Professionalのフィルターパネルで設定可能なことを示します。○がついていないフィルターはActionScriptでのみ適用可能なものです。

DisplayObjectクラスのフィルター適用のためのプロパティ詳細

プロパティ	プロパティの詳細
filters :Array	表示オブジェクトに適用するフィルターオブジェクトが格納された配列です。適用するフィルターが1つの場合も配列に格納する必要があります。配列に2つ以上のフィルターオブジェクトが格納されている場合、配列の先頭から順に格納されたフィルターオブジェクトが累積的に適用されます。フィルターの適用を無効にする場合、このプロパティにnull値を設定します。

SAMPLE



※初期状態

SAMPLE CODE

DisplayObjectへのフィルターの追加 FilterSampleA.fla

```
import flash.display.*;
import flash.filters.*;
import flash.net.URLRequest;
import fl.controls.CheckBox;

//Loaderインスタンス変数
var imageLoader:Loader;
//フィルターインスタンス変数
var Bevel:BevelFilter;
var Blur:BlurFilter;                              ①
var DropShadow:DropShadowFilter;
var Glow:GlowFilter;

//初期処理関数
```

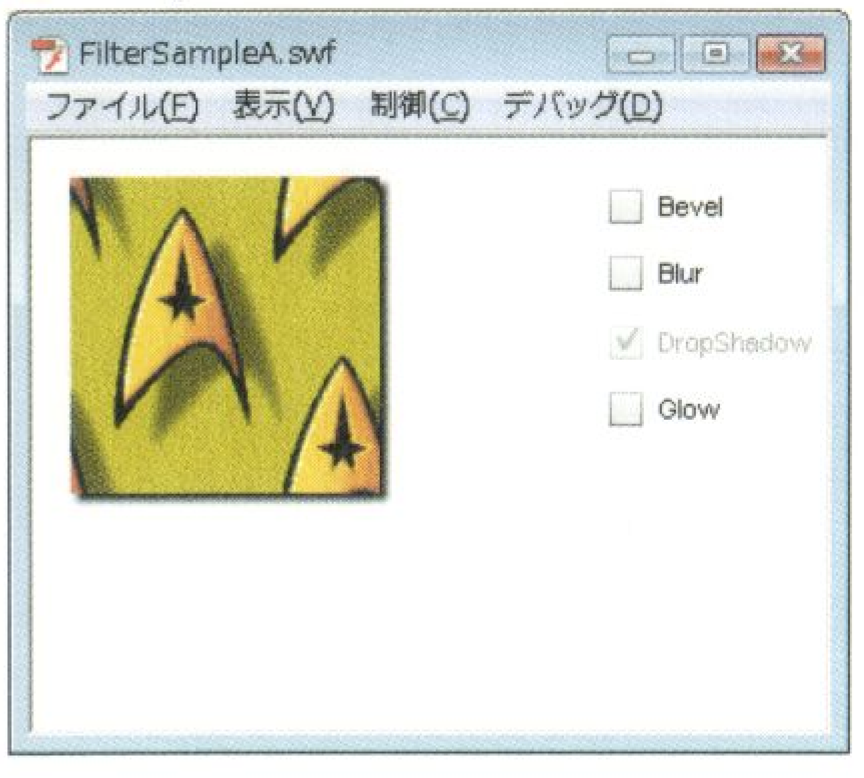
```
function init():void {
  //ステージに画像をロードして表示
  imageLoader = new Loader();
  var url:String = "images/sampleImage01.GIF";
  var urlReq:URLRequest = new URLRequest(url);
  imageLoader.load(urlReq);                                          ②
  imageLoader.x = 16;
  imageLoader.y = 16;
  addChild(imageLoader);
  //フィルターオブジェクトの作成
  Bevel = new BevelFilter();
  Blur = new BlurFilter();                                           ③
  DropShadow = new DropShadowFilter();
  Glow = new GlowFilter();
  //チェックボックスのイベントリスナーの登録                              ④
  this.Bevel_chk.addEventListener(Event.CHANGE, setFilter );
  this.Blur_chk.addEventListener(Event.CHANGE, setFilter );
  this.DropShadow_chk.addEventListener(Event.CHANGE, setFilter );
  this.Glow_chk.addEventListener(Event.CHANGE, setFilter );
}

//フィルターをセットするハンドラ関数
function setFilter(event:Event):void { ―――――――――――――――――――――― ⑤
  //フィルター配列変数に既存の配列値を代入
  var newFilters:Array = imageLoader.filters; ――――――――――――――――― ⑥
  if ( ( event.target as CheckBox ).selected ) {
    //フィルター配列にフィルターオブジェクトを追加
    newFilters.push( this[ event.target.label ]  ); ――――――――――――― ⑦
    //チェックボックスを変更できないようにする
    event.target.enabled = false;
  }
  //表示オブジェクトにフィルター配列を設定
  imageLoader.filters = newFilters; ―――――――――――――――――――――――― ⑧
}
init();
```

DisplayObjectへのフィルター適用手順に従ってフィルターを設定するサンプルです。適用先の画像は②で、Loaderオブジェクトを使用してファイルシステムから読み込みます。③で

フィルターオブジェクトを作成していますが、何も設定を行わないため、適用されるフィルターはデフォルト値によるものになります。④でチェックボックスのイベントハンドラを設定していますが、実際にフィルターを設定するのが⑤からのハンドラ関数です。⑥で既存のフィルター配列をnewFilters:Array変数に退避し、⑦でその変数にフィルターオブジェクトを追加しています。この行のthis[event.target.label]という記述は、チェックボックスのラベルに①で宣言しているフィルター変数と同じ名前がつけられているため、それを利用してフィルターオブジェクトを参照しています。⑧でフィルター配列を再度設定していますが、これでフィルターが適用されます。

■ DropShadowFilterを追加



■ BlurFilterを追加



8-2-2 フィルターをBitmapData オブジェクトに適用する

8-2-1で解説したように、flash.filtersパッケージのフィルタークラスオブジェクトを作成して、DisplayObjectまたはBitmapDataオブジェクトにフィルターを適用することができます。BitmapDataオブジェクトにフィルターを適用する場合、下記のようにフィルターの適用先と結果が別オブジェクトになるため、元のオブジェクトはそのままの状態を保てることと、対象のビットマップの適用範囲を指定できるのが特徴的です。ActionScript3.0で使用できるフィルターについては、前節をご覧ください。

BitmapDataへのフィルター適用手順

順番	処理内容
1	適用するフィルターのインスタンスを作成し、プロパティを設定する。
2	BitmapDataオブジェクトのapplyFilter()メソッドのパラメータにフィルターを適用するBitmapDataオブジェクト（適用先）と1で作成したフィルターインスタンス、適用範囲などを指定する。結果はメソッドを呼び出したオブジェクトに適用され、適用先のオブジェクトはそのまま残ります。
3	累積的にフィルターを適用するには、1と2の処理を繰り返します。

BitmapDataクラスのフィルター適用のためのメソッド詳細

メソッド名	メソッドの詳細
`applyFilter( sourceBitmapData: BitmapData, sourceRect:Rectangle, destPoint:Point, filter:BitmapFilter ):void`	**sourceBitmapData**：フィルターを適用するBitmapDataオブジェクトを指定します。 **sourceRect**：sourceBitmapDataオブジェクト内のフィルターを適用する領域を指定します。 **destPoint**：sourceRectで指定した領域の左上位置を現在のBitmapDataオブジェクト内のどの位置に適用するかを指定します。 **filter**：適用するフィルターオブジェクトです。 メソッドにより、sourceBitmapDataで指定したオブジェクトに、sourceRectおよびfilterによってフィルターを適用した結果が現在のBitmapDataオブジェクトのdestPointを基点とする位置に適用されます。

SAMPLE

■ Bevelを適用



■ DropShadowを適用



■ Blurを適用



■ Glowを適用



SAMPLE CODE

BitmapDataオブジェクトにフィルターを適用する

FilterSampleB.fla

```
import flash.display.*;
import flash.filters.*;
import flash.net.URLRequest;
import fl.controls.CheckBox;

//Loaderインスタンス変数
var imageLoader:Loader;
//フィルターインスタンス変数
var Bevel:BevelFilter;
var Blur:BlurFilter;
var DropShadow:DropShadowFilter;
var Glow:GlowFilter;
```

```
//applyFilter()メソッドで使用するパラメータ用変数
import flash.geom.Rectangle;
import flash.geom.Point;
//適用先のBitmapData
var target:BitmapData;
//BitmapData表示用Bitmap
var targetBitmap:Bitmap;

//初期処理関数
function init():void {
  //ステージに画像をロードして表示
  imageLoader = new Loader();
  var url:String = "images/sampleImage01.GIF";
  var urlReq:URLRequest = new URLRequest(url);
  imageLoader.load(urlReq);
  imageLoader.x = 16;
  imageLoader.y = 16;
  addChild(imageLoader);
  //フィルターオブジェクトの作成
  Bevel = new BevelFilter();
  Blur = new BlurFilter();
  DropShadow = new DropShadowFilter();
  Glow = new GlowFilter();
  //チェックボックスのイベントハンドラ
  this.Bevel_chk.addEventListener(Event.CHANGE, setFilter );
  this.Blur_chk.addEventListener(Event.CHANGE, setFilter );
  this.DropShadow_chk.addEventListener(Event.CHANGE, setFilter );
  this.Glow_chk.addEventListener(Event.CHANGE, setFilter );
  //適用先のBitmapDataオブジェクトを作成
  target = new BitmapData( 140, 140,  true, 0xFFFFFFFF );   ┐
  targetBitmap = new Bitmap( target );                       │
  targetBitmap.x = 170;                                      ├─①
  targetBitmap.y = 11;                                       │
  //Bitmapオブジェクトを表示リストに追加                       │
  addChild( targetBitmap );                                  ┘
}

//フィルターを適用するハンドラ関数
function setFilter(event:Event):void { ──────────②
```

```
    //前回のフィルター効果をクリアするために矩形を塗りつぶす
    target.fillRect( new Rectangle( 0, 0, 140, 140 ), 0xFFFFFFFF );  ③
    if ( ( event.target as CheckBox ).selected ) {
      //フィルターの適用
      target.applyFilter( ( imageLoader.content as Bitmap ).
      bitmapData, new Rectangle( 0, 0, 128, 128 ), new Point( 5, 5 ),
      this[ event.target.label ] );                                  ④
      //チェックボックスを変更できないようにする
      event.target.enabled = false;
    }
  }

  init();
```

8-2-1のFilterSampleA.flaを流用して、フィルターを適用する個所を変更しています。このサンプルでは、累積的なフィルターの適用は行わずに、毎回新たに適用しています。BitmapDataオブジェクトを使用する場合、適用元、適用先の2オブジェクトを利用することになります。まず適用元の画像は、8-2-1と同じようにLoaderオブジェクトでロードします。適用先のオブジェクトは、①で表示リストへの追加まで行っています。②からのseetFilter()関数でBitmapDataオブジェクトへのフィルターの適用を行っています。③でfillRect()メソッドを実行しているのは、前回のフィルター適用後の画像があった場合、クリアするためです。④でapplyFilter()メソッドを使用しており、適用元の画像のサイズは128×128ですが、140×140のサイズをとっています。これは、フィルター適用後に、その効果によって画像のサイズが大きくなることがあるからです。

8-3-1 グラフィック描画APIを使う

ActionScriptで直線、曲線、矩形、円、塗り、グラデーションなどのグラフィック図形描画を行うには、Graphicsクラスを使用します。Shape、Sprite、MovieClipクラスのインスタンス上に描画することができます。これらのクラスにはgraphicsというプロパティが用意されており、Graphicsオブジェクトを生成する必要はなく、このプロパティを通してGraphicsオブジェクトを利用できます。つまり、「描画対象オブジェクト.graphics.Graphicsクラスメソッド() 」という呼び出しを行って描画処理を行うことができます。描画を開始する前に、色や塗りなどのスタイルを決定するメソッドを呼び出しておく必要があります。グラフィック描画を行う場合、Shape、Sprite、MovieClipのどのクラスインスタンスを使用するかについては、下記の表「Graphicsクラスによる描画対象のクラス」を参考にしてください。サンプルコードでは、もっとも手軽に円や四角形を描画することができる、シェイプ描画メソッドを使用します。シェイプ描画メソッドはビルトインメソッドとも呼ばれ、下表「Graphicsクラスの主な描画メソッド概要(シェイプ描画)」のものがあります。

Graphicsクラスによる描画対象のクラス

クラス	備考
Shape	もっとも処理効率良くActionScriptによる描画が行えるインスタンスになります。ただし、DisplayObjectContainerを継承していないため、子を持つことができません。Shapeオブジェクトは描画を行うためのキャンパスということもできます。
Sprite	子を持つことができるオブジェクトとして、描画も行えます。タイムラインの操作を必要としない場合に使用します。
MovieClip	子を持つことができるオブジェクトとして、描画も行えます。タイムラインの操作を必要とする場合に使用します。

Graphicsクラスの主な描画メソッド概要(シェイプ描画)

目的	メソッド名	メソッドの概要
円を描画する	`drawCircle(` `x:Number,` `y:Number,` `radius:Number` `):void`	(x, y)座標を中心、radiusを半径(ピクセル)とする円を描画します。
楕円を描画する	`drawEllipse (` `x:Number,` `y:Number,` `width:Number,` `height:Number` `):void`	(x, y)座標を境界ボックスの左上、widthとheightを幅と高さとする楕円を描画します。

Graphicsクラスの主な描画メソッド概要（シェイプ描画）

目的	メソッド名	メソッドの概要
矩形を描画する	`drawRect(` `x:Number,` `y:Number,` `width:Number,` `height:Number` `):void`	(x, y)座標を左上、widthとheightを幅と高さとする矩形を描画します。
角丸の矩形を描画する	`drawRoundRect(` `x:Number,` `y:Number,` `width:Number,` `height:Number,` `ellipseWidth:Number,` `ellipseHeight:Number = NaN` `):void`	(x, y)座標を左上、widthとheightを幅と高さ、ellipseWidth、ellipseHeightをコーナーに適用する楕円の幅と高さとする角丸矩形を描画します。

Graphicsクラスの主な描画メソッド概要（塗りと線スタイル）

目的	メソッド名	メソッドの概要
ビットマップデータで塗りつぶす	`beginBitmapFill(` `bitmap:BitmapData,` `matrix:Matrix = null,` `repeat:Boolean = true,` `smooth:Boolean = false` `):void`	描画領域をビットマップイメージで塗りつぶします。
描画する色を指定する	`beginFill(` `color:uint,` `alpha:Number = 1.0` `):void`	lineTo()やdrawCircle()などを行う前に、塗りの色を指定します。
グラデーション塗りを指定する	`beginGradientFill(` `type:String,` `colors:Array,` `alphas:Array,` `ratios:Array,` `matrix:Matrix = null,` `spreadMethod:String = "pad",` `interpolationMethod:String = "rgb",` `focalPointRatio:Number = 0` `):void`	Graphicsオブジェクトの後続の呼び出しに使用するグラデーション塗りを指定します。
シェーダ塗りを指定する	`beginShaderFill(` `shader:Shader,` `matrix:Matrix = null` `):void`	Graphicsオブジェクトの後続の呼び出しに使用するシェーダ塗りを指定します。
塗りと線のスタイルをクリアする	`clear():void`	Graphicsオブジェクトに描画されているグラフィックをクリアして塗りと線のスタイル設定をリセットします。

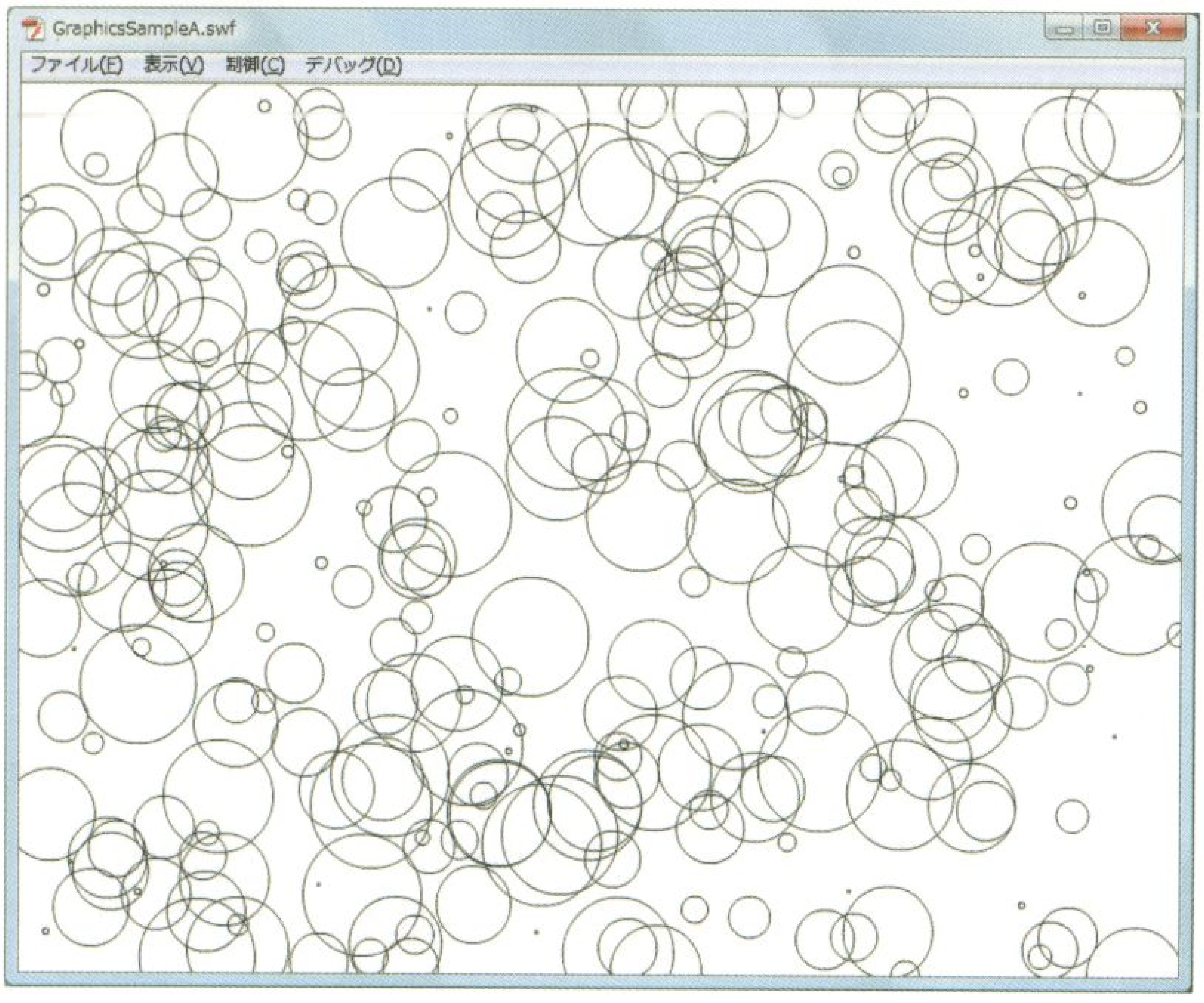
Graphicsクラスの主な描画メソッド概要（塗りと線スタイル）（続き）

目的	メソッド名	メソッドの概要
線のグラデーションを指定する	`lineGradientStyle(` `type:String,` `colors:Array,` `alphas:Array,` `ratios:Array,` `matrix:= null,` `spreadMethod:String = "pad",` `interpolationMethod:String = "rgb",` `focalPointRatio:Number = 0` `):void`	線の描画で使用するグラデーションを指定します。
線のスタイルを指定する	`lineStyle(` `thickness:Number = NaN,` `color:uint = 0,` `alpha:Number = 1.0,` `pixelHinting:Boolean = false,` `scaleMode:String = "normal",` `caps:String = null,` `joints:String = null,` `miterLimit:Number = 3` `):void`	lineTo()メソッドやdrawCircle()メソッドなど、Graphicsのメソッドの後続の呼び出しに使用する線スタイルを指定します。

Graphicsクラスの主な描画メソッド概要（直線と曲線の描画）

目的	メソッド名	メソッドの概要
曲線を描画する	`curveTo(` `controlX:Number,` `controlY:Number,` `anchorX:Number,` `anchorY:Number` `):void`	現在の描画位置から（anchorX, anchorY）まで、（controlX, controlY）で指定されたコントロールポイントを使用して曲線を描画します。
ビットマップを使用して線を描く	`lineBitmapStyle(` `bitmap:BitmapData,` `matrix:Matrix = null,` `repeat:Boolean = true,` `smooth: Boolean = false` `):void`	線として使用するビットマップを指定します。
線を描画する	`lineTo(` `x:Number,` `y:Number` `):void`	現在の描画位置から(x, y)まで、現在の線のスタイルを使用して線を描画します。その後で、現在の描画位置は(x, y)に設定されます。
描画位置を指定する	`moveTo(` `x:Number,` `y:Number` `):void`	描画位置を(x, y)に移動します。

SAMPLE



SAMPLE CODE

Graphicsクラスによるシェイプの描画

GraphicsSampleA.fla

```
import flash.display.Shape;

//描画を行うためのShapeオブジェクト変数
var campus:Shape;
//円のサイズを表す変数
var circleSize:uint;

//ランダムな円を描く関数
function drawShapes():void { ──────────────── ①
  for (var i:int = 1; i <= 300; i++) {
    //Shapeオブジェクトを作成
    campus = new Shape();
    //ステージの高さ/14を最大とするサイズをランダムに決定
    circleSize =Math.random()*this.stage.stageHeight/14; ──── ②
    //線を1ピクセル黒に設定
    campus.graphics.lineStyle( 1, 0x000000 ); ──────── ③
    //円を描く
    campus.graphics.drawCircle(0, 0, circleSize);
```

```
        campus.graphics.endFill();
        //Shapeの位置をステージ内でランダムに配置
        campus.x=Math.random()*this.stage.stageWidth;     ④
        campus.y=Math.random()*this.stage.stageHeight;
        //表示リストに追加
        addChild(campus);     ⑤
    }
}

drawShapes();
```

drawCircle()メソッドを使用するサンプルです。①からのループで300個の円を描画します。②でステージサイズをもとにしてcircleのサイズをランダムに求めて③で描画します。④でShapeの位置をステージ内でランダムに配置することにより結果イメージのような描画が行われます。⑤で行っているように、描画を行ったShapeオブジェクトはaddChilde()メソッドで表示リストに追加する必要があります。

8-3-2 線と塗りを定義する

8-3-1 のサンプルコードでは、黒一色で描画しましたが、実際には好みの線と塗りを設定してから描画を行います。グラフィックAPI描画時の手順は下記「グラフィックAPIによる描画手順」のとおりです。特徴として、一度指定した線と塗りの設定は保持されるため、線と塗りを変更する場合は5番にあるように再設定を行いますが、そうでない場合は同じ設定で複数の描画を継続的に行うことができます。ここでは、単色の線と塗りについて解説します。グラデーションによる線と塗りについては、8-3-4「グラデーションの描画を行う」をご覧ください。

グラフィックAPIによる描画手順

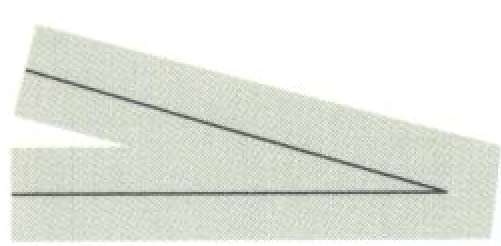
順番	処理内容
1	graphicsプロパティを持つShape、Sprite、MovieClipオブジェクトを作成する
2	線と塗りを定義する
3	描画する位置を決める
4	描画APIによる描画を行う
5	別の線と塗りで描画する場合、現在の塗りを終了して再設定を行う

Graphicsクラスの単色による線と塗り設定のためのメソッド詳細

メソッド名	メソッドの詳細
`beginFill( color:uint, alpha:Number = 1.0 ):void`	**color**：描画メソッドに対する今後の呼び出しに使用する一色塗りを指定します。 **alpha**：アルファ透明度を指定します。指定しない場合、1.0（透明度なし）です。 beginFill()、beginBitmapFill()、beginGradientFill()を次に呼び出すまで、その一色塗りは継続します。
`clear():void`	Graphicsオブジェクトに描画されているグラフィックをクリアし、塗りと線のスタイルの設定をリセットします。
`endFill():void`	beginFill()、beginGradientFill()、beginBitmapFill()メソッドによって設定された塗りをそれまでの描画オブジェクトに適用します。例えば描画した線が閉じられていない場合、このメソッドによりパスを閉じて塗りが行われます。

PART
1
2
3
4
5
6
7
8
9

Graphicsクラスの単色による線と塗り設定のためのメソッド詳細（続き）

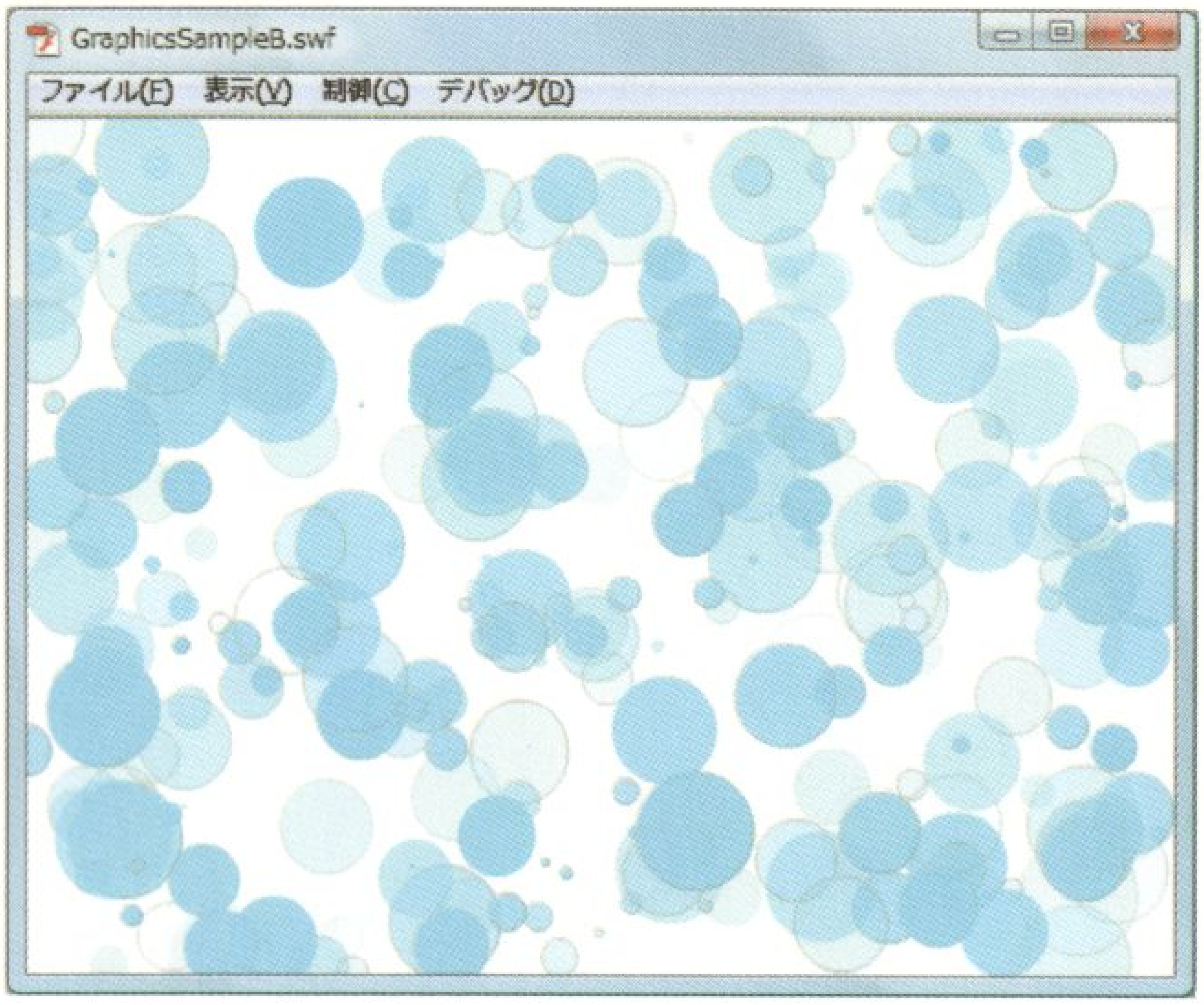
メソッド名	メソッドの詳細
lineStyle(thickness:Number = NaN, color:uint = 0, alpha:Number = 1.0, pixelHinting:Boolean = false, scaleMode:String = "normal", caps:String = null, joints:String = null, miterLimit:Number = 3):void	線スタイルを指定します。drawRect()などのシェイプ描画メソッドやlineTo()などの線描画メソッドで使用する線に適用されます。 再びこのメソッドやlineBitmapStyle()、lineGradientStyle()メソッドを呼び出すまで設定が継続されます。 **thickness**：線の太さを0～255のポイント単位で指定します。0はデフォルト値で、極細線になります。 **color**：線の色を16進数で指定します。 **alpha**：線のアルファ値を0～1の範囲で指定します。 **pixelHinting**：trueの場合、線のピクセルヒンティングを行います。 **scaleMode**：描画オブジェクトが拡大/縮小したときの線の太さの拡大/縮小を決めるモードです。 **LineScaleMode.NORMAL**：オブジェクトの拡大/縮小とともに線の太さも拡大/縮小します。 **LineScaleMode.NONE**：線の太さは拡大/縮小しません。 **LineScaleMode.VERTICAL/LineScaleMode.HORIZONTAL**：描画オブジェクトを垂直/水平方向のみに拡大/縮小する場合、線の太さを拡大/縮小しません。 **caps**：線の終端の形状（キャップ）を指定します。 **CapsStyle.NONE**：キャップを描画しません。 **CapsStyle.ROUND**：丸いキャップを使用します。 **CapsStyle.SQUARE**：四角いキャップを使用します。  **joints**：三角形の頂点のような角を描くときの形状を指定します。 **JointStyle.MITER**：マイター形状を描画します。 **JointStyle.ROUND**：丸角の形状を描画します。 **JointStyle.BEVEL**：ベベルの形状を描画します。  **miterLimit**：jointStyle=JointStyle.MITERのとき、マイターが切り取られる限度を1～255の範囲で指定します。 下図はマイターが切り取られたイメージです。 

Graphicsクラスの単色による線と塗り設定のためのメソッド詳細（続き）

メソッド名	メソッドの詳細
	マイターが切り取られる、コーナーの最大角度は下表のとおりです。最大角度以下の場合、切り取られることになります。

miterLimit	最大角度
1.414	90度
2	60度
4	30度
8	15度

SAMPLE



SAMPLE CODE

線と塗りを変化させて描画する

GraphicsSampleB.fla

```
import flash.display.Shape;

//描画を行うためのShapeオブジェクト変数
var campus:Shape;
//背景色の最大値
var bgColor:uint=0x65CCFF;
//円のサイズ
var circleSize:uint;
//ボーダ色の最大値
```

```
var borderColor:uint=0xaaaaaa;
//ボーダサイズの最大値
var borderSize:uint=1.3;

//ランダムな円を描く関数
function drawShapes():void {
  for (var i:int = 1; i <= 300; i++) {
    campus = new Shape();
    //ステージの高さ/14を最大とするサイズをランダムに決定
    circleSize=Math.random()*this.stage.stageHeight/14;
    //塗りの色を指定して、アルファをランダムに決定
    campus.graphics.beginFill(bgColor, Math.random
    () ); ―――――――――――――――――――――――――――――――― ①
    //ボーダのサイズとアルファをランダムに決定
    campus.graphics.lineStyle(Math.random()*borderSize, borderColor,
    Math.random()); ――――――――――――――――――――――――― ②
    //円を描く
    campus.graphics.drawCircle(0, 0, circleSize);
    campus.graphics.endFill();
    //シェイプの位置をランダムに決定
    campus.x=Math.random()*this.stage.stageWidth;
    campus.y=Math.random()*this.stage.stageHeight;
    //表示リストに追加
    addChild(campus);
  }
}

drawShapes();
```

8-3-1 のGraphicsSampleB.flaサンプルに修正を加えて線と色をある範囲内でランダムに変化させるようにしています。①のbeginFill()メソッドの第2パラメータでalpha値をランダムに指定しています。Math.random()メソッドは0.0〜1.0の範囲で値を返すため、そのままahpha値に使用しています。②のlineStyle()メソッドでは、thicknessを最大1.3ポイント、alpha値を同様にMath.random()で設定し、上記のイメージを描画しています。

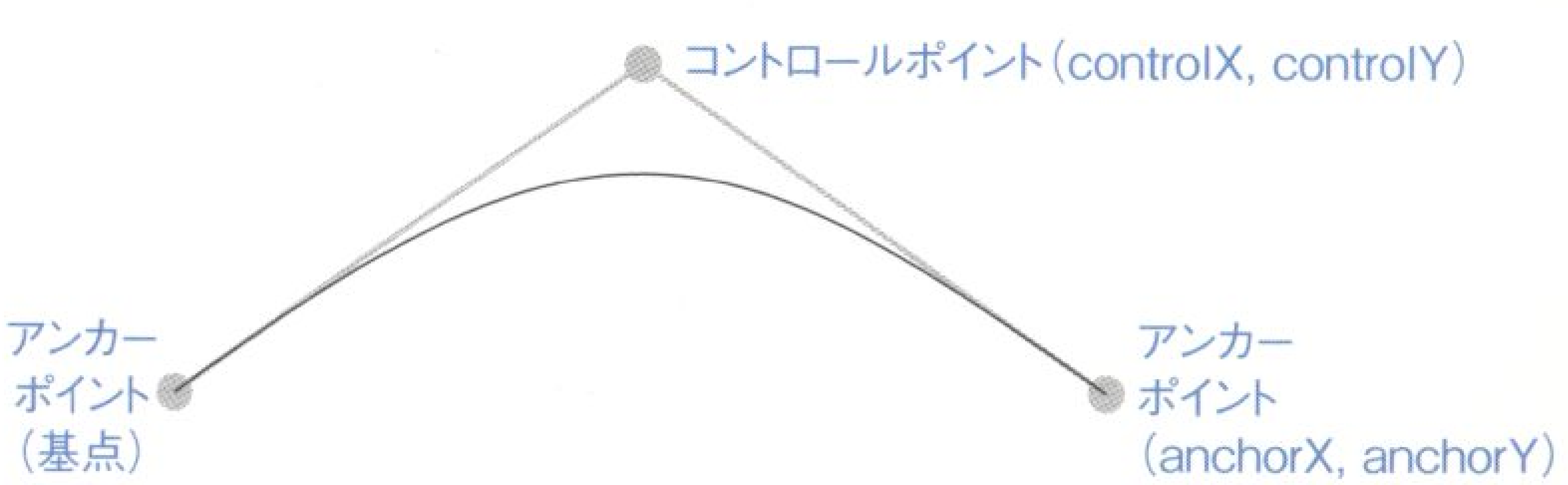
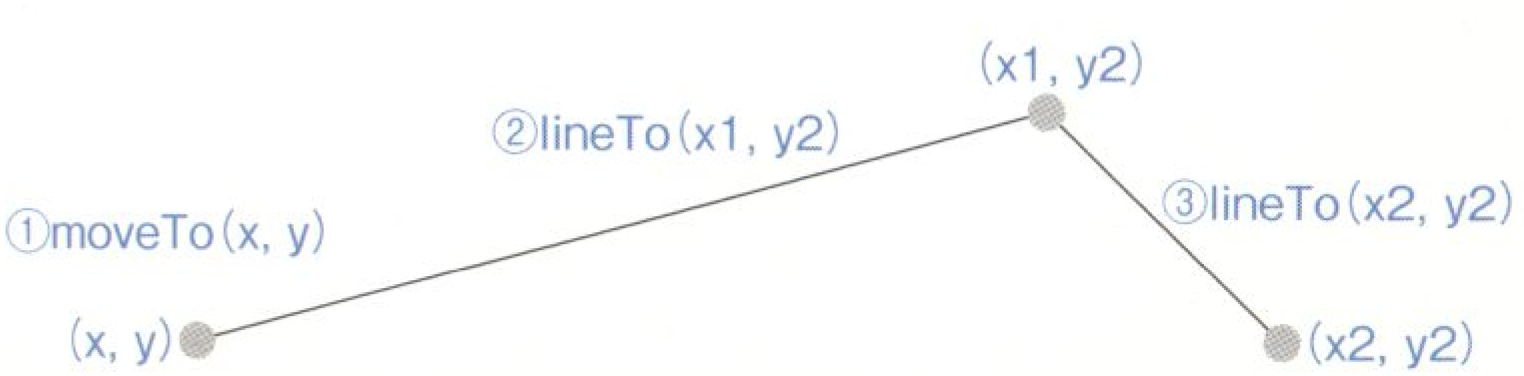
8-3-3 線と曲線を描画する

ActionScriptでは、描画の基本となる線や曲線を描くメソッドが提供されます。drawCircle()、drawRect()などのシェイプ描画メソッドもこれらの基本メソッドを利用して矩形や円を描きます。描画するときは必ず「現在の位置」を基準にして線を描きます。単純に現在の位置を移動させるためにmoveTo()というメソッドが用意されています。曲線を描く場合、2つのアンカーポイントと1つのコントロールポイントで構成される二次ベジェ曲線による描画を行います。線や曲線を描画する手順は下表のとおりです。

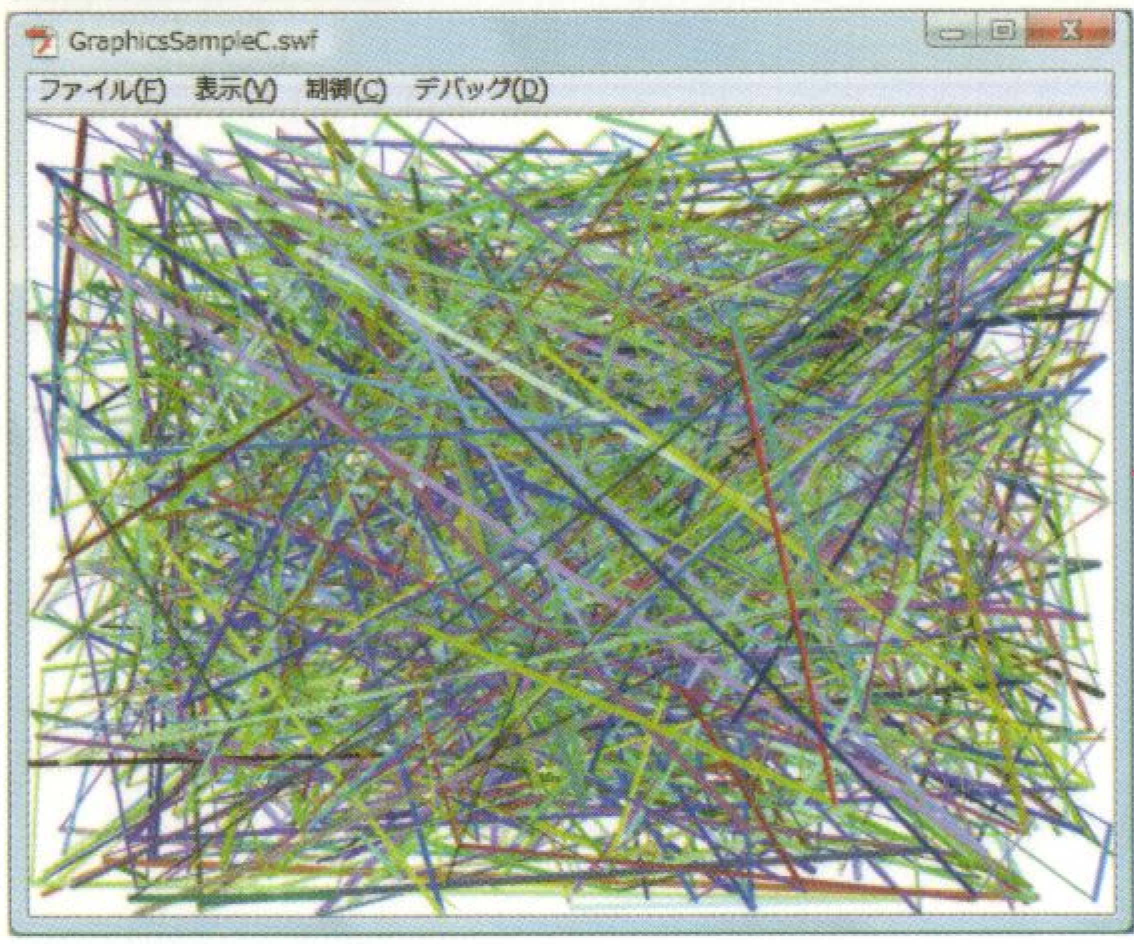
線や曲線の基本描画手順

順番	処理内容
1	graphicsプロパティを持つShape、Sprite、MovieClipオブジェクトを作成する
2	線と塗りを定義する
3	moveTo()メソッドで描画の基点を設定する
4	lineTo()、curveTo()による直線、曲線の描画を行う
5	別の線と塗りで描画する場合、現在の塗りを終了して再設定を行う

Graphicsクラスの線と塗り設定のためのメソッド詳細

メソッド名	メソッドの詳細
moveTo(x:Number, y:Number):void	描画位置を(x, y)に移動します。この位置からcurveTo()、lineTo()を実行して曲線、直線を描画します。
curveTo(controlX:Number, controlY:Number, anchorX:Number, anchorY:Number):void	描画位置から(controlX, controlY)のコントロールポイントを使用して、アンカーポイント(anchorX, anchorY)まで曲線を描画します。 
lineTo(x:Number, y:Number):void	描画位置から(x, y)まで、線を描画します。描画後現在の描画位置は(x, y)になります。 

SAMPLE



lineTo()で様々な色と太さの直線を描画

SAMPLE CODE

線と曲線の描画

GraphicSampleC.fla

```
import flash.display.Shape;
//描画を行うためのShapeオブジェクト変数
var campus:Shape;
//線色の最大値
var lineColor:uint=0xBBBBBB;
//線の太さの最大値
var lineThickness:uint=4;
//ランダムな線を描く関数
function drawShapes():void {
  campus = new Shape();
  campus.graphics.moveTo( 0, 0 );
  for (var i:int = 1; i <= 1000; i++) {
    //太さと色をランダムに設定
    campus.graphics.lineStyle( Math.random()*lineThickness,
        Math.random()*lineColor );
    //ステージの高さ/14を最大とするサイズをランダムに決定
    campus.graphics.lineTo( Math.random()*this.stage.stageWidth,
         Math.random()*this.stage.stageHeight );
  }
  //表示リストに追加
  addChild(campus);
}
drawShapes();
```

①ループ内で1000回lineTo()を実施

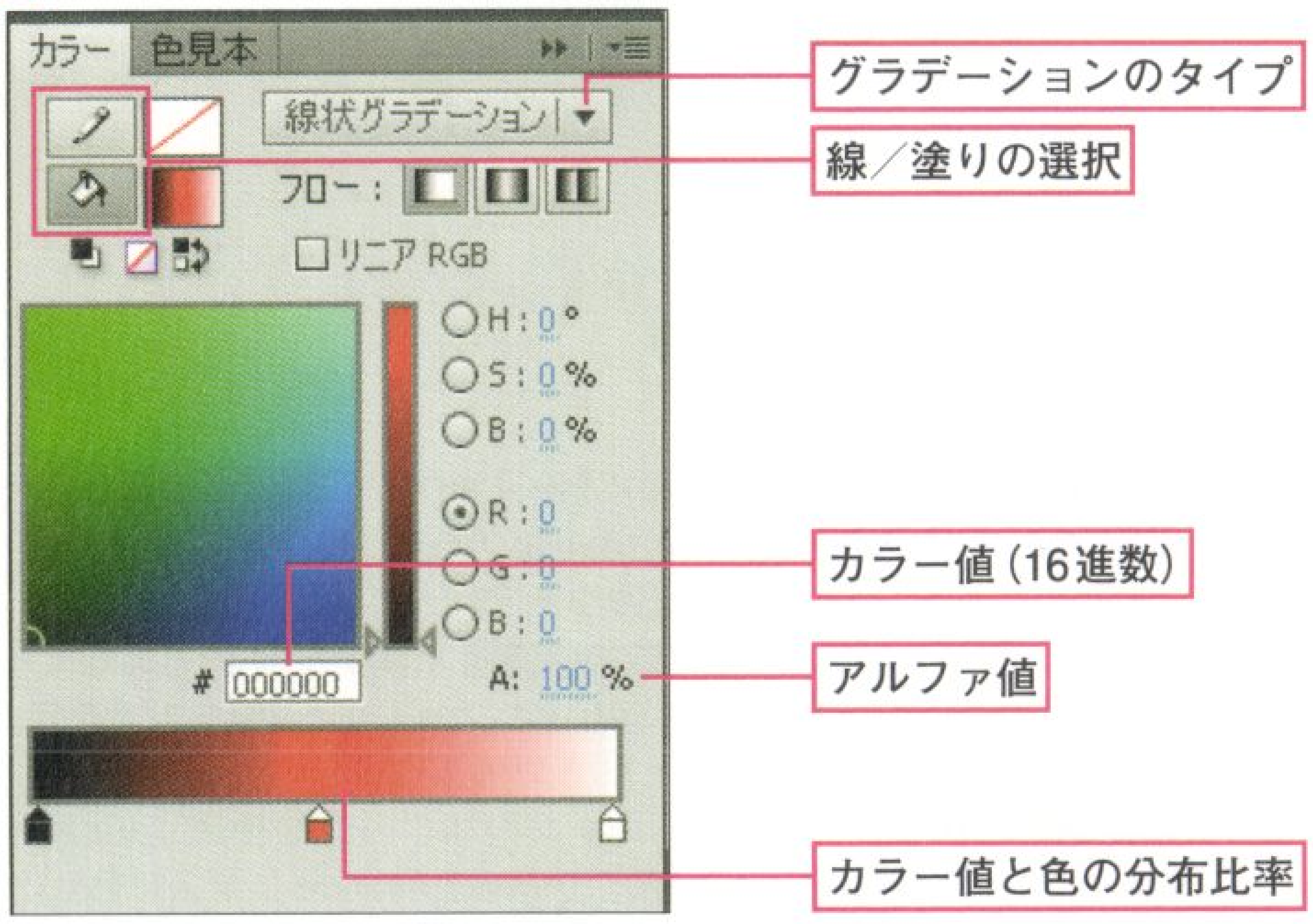
②太さと色をランダムに設定

③長さ(終点)をランダムに設定

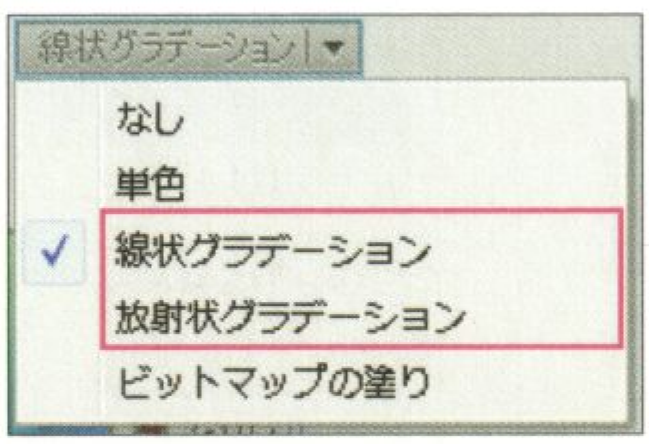
8-3-4 グラデーションの描画を行う

線と塗りは単色だけでなくグラデーションで行うことができます。下表にあるように、線のグラデーションを設定するには、lineGradientStyle()メソッド、塗りのグラデーションはbeginGradientFill()メソッドで行います。これらのメソッドのパラメータは多いですが、パラメータの設定値はオーサリングツールの「カラー」パネルと対応している部分が多く、そちらを参考にすると理解が深まります。

■ Flash CS5のカラーパネル

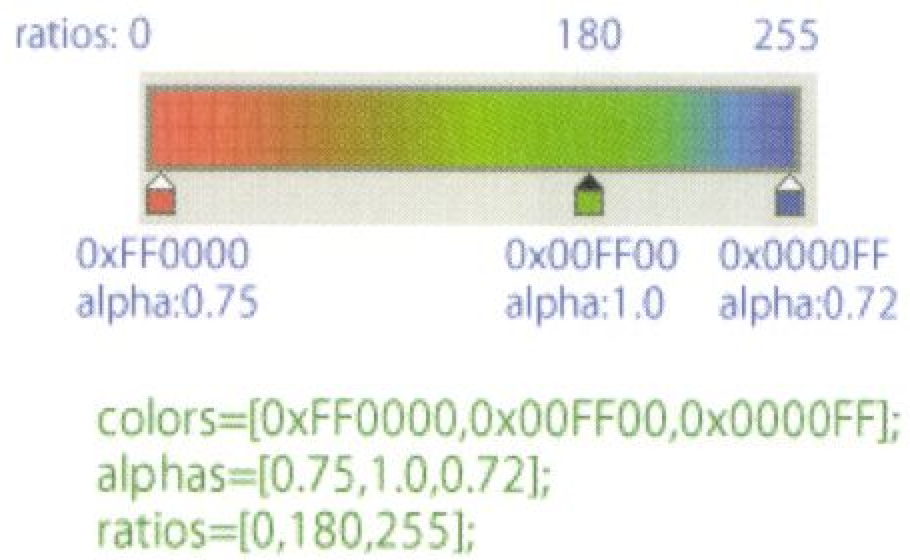
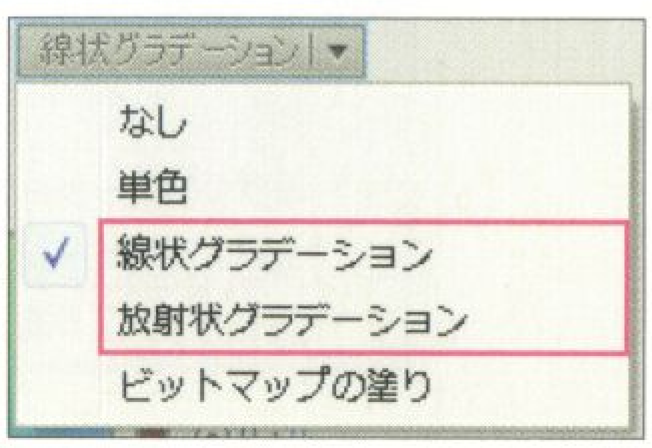


Graphicsクラスのグラデーション線と塗り設定のためのメソッド詳細

メソッド名	メソッドの詳細
beginGradientFill(type:String, colors:Array, alphas:Array, ratios:Array, matrix:Matrix = null, spreadMethod:String = "pad", interpolationMethod:String = "rgb", focalPointRatio:Number = 0):void	lineTo()、curveTo()、drawCircle()をはじめとするシェイプ描画メソッドの塗りで使用するグラデーションを指定します。設定は次に再びこのメソッドやbeginFill()、beginBitmapFill()、beginShaderFill()などの塗り設定メソッドを呼び出すまで保持されます。 **type**：描画メソッドに対する今後の呼び出しに使用するグラデーション塗りのタイプを指定します。 **GradientType.LINEAR**：線状パスにブレンドするグラデーション **GradientType.RADIAL**：中央の焦点から外側に向かって放射状にブレンドするグラデーション 

PART
1
2
3
4
5
6
7
8
9

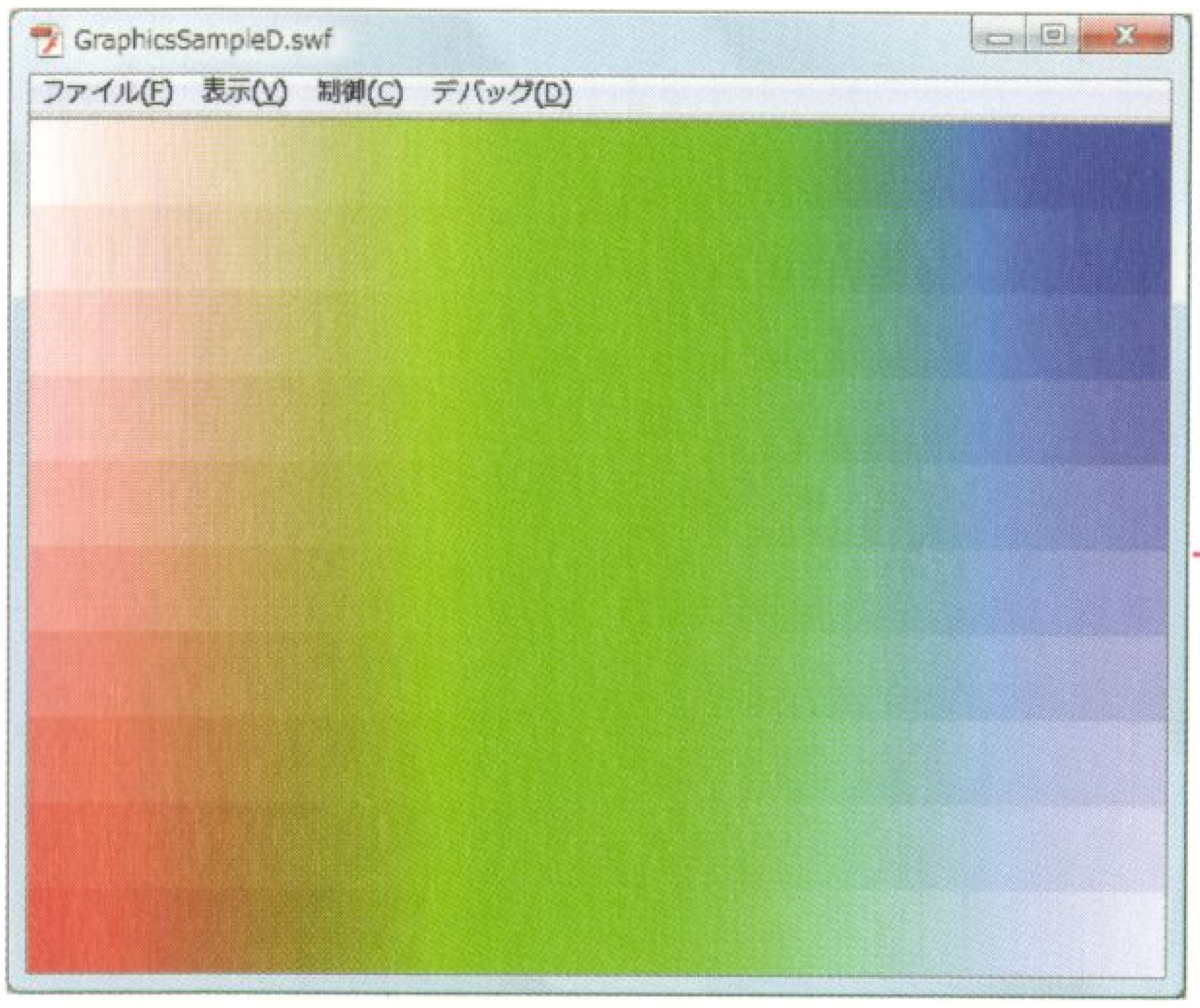
Graphicsクラスのグラデーション線と塗り設定のためのメソッド詳細（続き）

メソッド名	メソッドの詳細
	colors：グラデーションで使用する色を最大15色まで配列で指定します。 alphas：colors配列の色と対応するアルファ値を配列で指定します。 **ratios**：colors配列の色の分布比率を配列で指定します。0を左端、255を右端とする位置を使用して、colors/alphasで設定したそれぞれの色とアルファ値の相対位置を表します。下図はcolors、alphas、ratiosのサンプル値をFlashのカラーパネルの一部を引用して表したものです。  **matrix**：グラデーション用のMatrixオブジェクトを作成して適用できます。flash.geom.MatrixクラスのcreateGradientBox()メソッドで作成した変換マトリックスを指定できます。グラデーション塗りの幅と高さは描画オブジェクトではなくこのパラメータで指定したMatrixオブジェクトが持つグラデーションマトリックスの幅と高さで決まります。 **spreadMethod**：グラデーションのパターンを決定するspreadメソッドを指定できます。SpreadMethodクラスにPAD、REFLECT、REPEATの3種類の定数が用意されています。 **interpolationMethod**：グラデーションをレンダリングするときに使用するRGBスペースを決定します。 InterpolationMethod.LINEAR_RGB（線形RGB補間メソッドを使用）、InterpolationMethod.RGB（RGB補間メソッドを使用：デフォルト）を選択できます。 **focalPointRatio**：グラデーションの焦点を制御する数値を−1～1の範囲で指定します。
`lineGradientStyle(` `type:` `colors:Array,` `alphas:Array,` `ratios:Array,` `matrix:Matrix = null,` `spreadMethod:String = "pad",` `interpolationMethod:String =` `"rgb",` `focalPointRatio:Number = 0` `):void`	lineTo()、curveTo()、drawCircle()をはじめとするシェイプ描画メソッドの線で使用するグラデーションを指定します。設定は次に再びこのメソッドやlineStyle()などのメソッドを呼び出すまで保持されます。描画の前にlineStyle()メソッドを使用してストロークの幅も指定しておく必要があります。 **type**：描画メソッドに対する今後の呼び出しに使用するグラデーション塗りのタイプを指定します。 GradientType.LINEAR、GradientType.RADIALを指定できます。 

Graphicsクラスのグラデーション線と塗り設定のためのメソッド詳細（続き）

メソッド名	メソッドの詳細
	colors：グラデーションで使用する色を最大15色まで配列で指定します。 **alphas**：colors配列の色と対応するアルファ値を配列で指定します。 **ratios**：colors配列の色の分布比率を配列で指定します。0を左端、255を右端とする位置を使用して、colors/alphasで設定したそれぞれの色とアルファ値の相対位置を表します。  colors=[0xFF0000,0x00FF00,0x0000FF]; alphas=[0.75,1.0,0.72]; ratios=[0,180,255]; **matrix**：グラデーション用のMatrixオブジェクトを作成して適用できます。flash.geom.MatrixクラスのcreateGradientBox()メソッドで作成した変換マトリックスを指定できます。 **spreadMethod**：グラデーションのパターンを決定するspreadメソッドを指定できます。SpreadMethodクラスにPAD、REFLECT、REPEATの3種類の定数が用意されています。 **interpolationMethod**：グラデーションをレンダリングするときに使用するRGBスペースを決定します。InterpolationMethod.LINEAR_RGB（線形RGB補間メソッドを使用）、InterpolationMethod.RGB（RGB補間メソッドを使用：デフォルト）を選択できます。 **focalPointRatio**：グラデーションの焦点を制御する数値を−1～1の範囲で指定します。

SAMPLE



よくあるFlashサンプルと同じようにRGB3色によるグラデーションを行ってますが、少しアレンジを加えてalphas配列を変化させています

SAMPLE CODE

beginGradientFill() によるグラデーション

GraphicsSampleD.fla

```
import flash.display.Shape;
//描画を行うためのShapeオブジェクト変数
var campus:Shape;
//colors配列
var colors:Array=[0xFF0000,0x00FF00,0x0000FF];
//alphas配列
var alphas:Array;
//ratios配列
var ratios:Array=[0,127,255];
//Matrixオブジェクト変数
var matrix:Matrix;
//グラデーション矩形を描く関数
function drawShapes():void {
  //描画オブジェクトの生成
  campus = new Shape();
  //Matrixオブジェクトの生成
  matrix = new Matrix();
  //グラデーションマトリックスオブジェクトの生成
  matrix.createGradientBox( this.stage.stageWidth, this.stage.
  stageHeight / 10,  0, 0, 0 );
  for (var i:int = 0; i < 10; i++) {
    //alpha値をループごとに変更
    alphas=[i/10,1.0,1-i/10];
    //beginGradientFill()の実行
    campus.graphics.beginGradientFill( GradientType.LINEAR, colors,
    alphas, ratios, matrix );
    //矩形を描く
    campus.graphics.drawRect( 0, i * (this.stage.stageHeight / 10),
    this.stage.stageWidth, this.stage.stageHeight / 10 );
    campus.graphics.endFill();
  }
  //表示リストに追加
  addChild(campus);
}
drawShapes();
```

①colors配列でRGB3色を決定

②一定の分布比率を定義

③ループで10回Rectを描画

④alphas配列をループインデックス変数で変化させる

PART 9

サウンドとビデオ



9-1-1 サウンドファイルをロードする

PART
1
2
3
4
5
6
7
8
9

ActionScriptで外部サウンドファイルを処理するには、まずFlash Playerにサウンドをロードする必要があります。サウンドリソースをロードする場合、リソースごとにSoundクラスのインスタンスを生成して処理を行います。SoundクラスはMP3形式のサウンドファイルをサポートします。サウンドリソースファイルをロードする場合、その場所を示すURLRequestインスタンスを作成して、Soundオブジェクトのload()メソッドに渡すとロードが開始されます。ロードが完了すると、SoundオブジェクトはEvent.COMPLETEイベントを送出してファイルは再生可能になります。コンストラクタでURLRequestオブジェクトを指定した場合、自動的にload()メソッドが呼び出されます。MP3ファイルに関しては、作者やジャンルなどを記録したID3タグと呼ばれるメタ情報も取得可能です。Soundクラスのイベントについては、9-1-2「サウンドのイベントを処理する」をご覧ください。Flash Playerが扱うサウンドデータはダウンロードするファイルのみではなく、外部マイク入力やストリーミングサーバからのサウンドデータも含まれます。Flash Playerがロードできるサウンドソース、サウンド処理を行うためのflash.mediaパッケージクラス、MP3のメタ情報を下表にまとめました。

Flash Playerでロード可能なサウンドソース

ソース	内容
外部サウンドファイル	実行時にロードしたファイル、またはSWFに埋め込んだサウンドファイル
マイク	ユーザ環境で利用可能なマイク入力
メディアサーバ	リモートメディアサーバーからストリーミングされるサウンド
動的に生成されたサウンド	Flash Playerが動的に作成したサウンド

flash.mediaパッケージのサウンド処理クラス

クラス	機能
Sound	サウンドのロード、サウンドプロパティの管理、サウンド再生の開始を行います。
SoundChannel	アプリケーション内でサウンドを制御します。サウンドはすべてサウンドチャンネルに割り当てられます。再生されるそれぞれのサウンドには、独自のSoundChannelオブジェクトがあり、サウンドの左再生チャンネルと右再生チャンネルのボリュームを制御します。アプリケーションがSoundオブジェクトを再生すると新しいSoundChannelオブジェクトが作成され、再生を制御できます。

flash.mediaパッケージのサウンド処理クラス(続き)

クラス	機能
SoundLoaderContext	ロードするファイルのセキュリティチェックを提供します。サウンドのロード中に何秒間バッファリングするか、また、ファイルのロード中にFlash PlayerまたはAIRがサーバーのポリシーファイルを検索するかどうかを指定します。SoundLoaderContextオブジェクトは、Sound.load()メソッドのパラメータで使用されます。
SoundMixer	アプリケーションの埋め込まれたストリーミングサウンドを制御します。共通のSoundMixerオブジェクトを通じて複数のサウンドチャンネルが合成されるので、SoundMixerオブジェクトのプロパティ値は、再生中のすべてのSoundChannelオブジェクトに影響を与えます。ActionScriptで生成されたサウンドは制御できません。
SoundTransform	ボリュームとパンを制御するプロパティを持ちます。SoundTransformオブジェクトは、SoundChannelオブジェクト、グローバルなSoundMixerオブジェクト、Microphoneオブジェクトなどに適用されます。
ID3Info	通常MP3サウンドファイルに格納されるID3メタデータ情報を表すプロパティが含まれています。Soundクラスのid3プロパティにアクセスして、MP3ファイルから追加のメタデータを取得できます。
Microphone	コンピュータに接続されたマイクやその他のサウンド入力デバイスからオーディオをキャプチャできます。マイクからのオーディオ入力はローカルのスピーカーに転送されるか、リモートサーバーに送信されます。Microphoneオブジェクトは、独自のサウンドストリームのゲイン、サンプリングレート、およびその他の性質を制御します。

Soundクラスのid3プロパティで参照可能なMP3メタデータ

ID3タグ	内容	参照プロパティ名(XXXXの部分)Sound.id3.XXXX
COMM	録音に関するコメント	comment/COMM
TALB	アルバム名	album/TALB
TCON	曲のジャンル	genre/TCON
TIT2	曲名	songName/TIT2
TPE1	アーティスト名	artist/TPE1
TRCK	トラック番号	track/TRCK
TYER	録音された年	year/TYER
TFLT	ファイル形式	TFLT
TIME	時刻	TIME
TIT1	内容の属するグループの説明	TIT1
TIT2	タイトル/曲名/内容の説明	TIT2
TIT3	サブタイトル/説明の追加情報	TIT3

1
1

■ Soundクラスのid3プロパティで参照可能なMP3メタデータ（続き）

ID3タグ	内容	参照プロパティ名（XXXXの部分）Sound.id3.XXXX
TKEY	最初の調	TKEY
TLAN	言語	TLAN
TLEN	長さ	TLEN
TMED	メディアタイプ	TMED
TOAL	オリジナルのアルバム/ムービー /ショーのタイトル	TOAL
TOFN	オリジナルのファイル名	TOFN
TOLY	オリジナルの作詞家/文書作成者	TOLY
TOPE	オリジナルのアーティスト/演奏者	TOPE
TORY	オリジナルのリリース年	TORY
TOWN	ファイルの所有者/ライセンス保持者	TOWN
TPE1	主な演奏者/ソリスト	TPE1
TPE2	バンド/オーケストラ/伴奏	TPE2
TPE3	指揮者/演奏者詳細情報	TPE3
TPE4	翻訳、リミックス、その他の修正を行った人	TPE4
TPOS	セット中の位置	TPOS
TPUB	発行者	TPUB
TRCK	トラック番号/セット内の位置	TRCK
TRDA	録音日	TRDA
TRSN	インターネットラジオ局の名前	TRSN
TRSO	インターネットラジオ局の所有者	TRSO
TSIZ	サイズ	TSIZ
TSRC	ISRC（国際標準録音資料コード）	TSRC
TSSE	エンコードに使用したソフトウェア/ハードウェアと設定	TSSE
TYER	年	TYER
WXXX	URLリンクフレーム	WXXX

※COMM ～ TYERのメタデータについては、ID3Infoクラスにプロパティとして定義されているため、プロパティ名が2つ存在することになり、どちらからも取得可能です。

Soundクラスの主なプロパティ概要

目的	プロパティ名	メソッドの概要
利用可能なバイト数を調べる	bytesLoaded:uint	[読み取り専用] Soundオブジェクトで現在利用できるデータのバイト数を表します。
オブジェクトの合計バイト数を調べる	bytesTotal:int	[読み取り専用] オブジェクトの合計バイト数を表します。
MP3ファイルのメタデータを調べる	id3:ID3Info	[読み取り専用] MP3ファイルが持つメタデータを表すID3Infoオブジェクトにアクセスできます。
MP3ファイルがバッファリングしているか調べる	isBuffering:Boolean	[読み取り専用] trueの場合、外部MP3ファイルがバッファリング中です。
サウンドの継続時間を調べる	length:Number	[読み取り専用] サウンドの継続時間をミリ秒数で表します。
サウンドがロードされるURLを調べる	url:String	[読み取り専用] サウンドがロードされるURLを表します。

Soundクラスの主なメソッド概要

目的	メソッド名	メソッドの概要
コンストラクタ	Sound(stream:URLRequest = null, context:SoundLoaderContext = null)	新しいSoundオブジェクトを作成します。streamにURLRequestオブジェクトを指定した場合、すぐにロードを開始します。
ストリームでダウンロードを中止する	close():void	ストリームを閉じてサウンドデータのダウンロードを中止します。
生のサウンドデータを抽出する	extract(target:ByteArray, lengthNumber, startPosition:Number = -1):Number	オブジェクトから生のサウンドデータを抽出します。
MP3ファイルをロードする	load(stream:URLRequest, context:SoundLoaderContext = null):void	streamで指定したURLから外部MP3ファイルのロードを開始します。
サウンドを再生する	play(startTime:Number = 0, loops:int = 0, sndTransform:SoundTransform = null):SoundChannel	サウンドを再生するSoundChannelオブジェクトを新しく作成します。

SAMPLE CODE

MP3ファイルのロードと再生

SoundSampleA.fla

```
//サウンドファイルをロードする関数
function loadSound():void {
  var soundUrl1:URLRequest = new URLRequest("sounds/SampleSound1.mp3"); ┐
  var sound1:Sound = new Sound();                                        ①┘
  //ロード完了時のリスナーの登録
  sound1.addEventListener( Event.COMPLETE, soundLoaded ); ――― ②
  sound1.load( soundUrl1 ); ――― ③
}

//ロードが完了した場合のイベントハンドラ
function soundLoaded(event:Event):void { ――― ④
  trace( (event.target as Sound).url + " ファイルがロードされました。" );
  //ターゲットサウンドを再生する
  (event.target as Sound).play(); ――― ⑤
  //ID3Infoオブジェクトを取得 ――― ⑥
  var id3:ID3Info = event.target.id3 as ID3Info;
  //ID3Infoクラスに定義されたプロパティを出力
  trace( "album:" + id3.album );          ┐
  trace( "artist:" + id3.artist );        │
  trace( "comment:" + id3.comment );      │
  trace( "genre:" + id3.genre );          ├― ⑦
  trace( "songName:" + id3.songName );    │
  trace( "track:" + id3.track );          │
  trace( "year:" + id3.year );            ┘
  //ID3Infoオブジェクトの全プロパティを出力
  trace( "--すべてのプロパティを出力します--" );
  for ( var id3Prop:String in id3 ) {        ┐
    trace( id3Prop + ":" + id3[ id3Prop ] ); ├― ⑧
  }                                          ┘
}

loadSound();
```

①でロードするリソース（ここではMP3ファイル）を示すURLRequestオブジェクトを作成しています。②では、ロードが完了したときのイベントハンドラを登録し、③でロードを開始します。④からがイベントハンドラですが、⑤でターゲットのサウンドオブジェクトに対して

play()メソッドを実行してロードしたサウンドを再生します。⑥以降はSoundオブジェクトのid3プロパティでアクセス可能なID3メタデータを出力しています。⑦では、ID3Infoクラスにプロパティとして定義済みのID3メタデータを出力します。⑧のforループですべてのメタデータを出力しています。MP3ファイルに定義されていないメタデータは出力されません。

▶ 実行結果（出力例）

```
file:///C|/ sounds/SampleSound1.mp3 ファイルがロードされました。
album:サンプルアルバム
artist:サンプルアーティスト
comment:コメントは特にありません。
genre:(12)
songName:Flash ActionScript3.0リファレンスサンプルサウンド1
track:null
year:2010-01-01
--すべてのプロパティを出力します--
TALB:サンプルアルバム
COMM:コメントは特にありません。
TIT2:Flash ActionScript3.0リファレンスサンプルサウンド1
TYER:2010-01-01
TCON:(12)
TPE1:サンプルアーティスト

--サウンドが再生されます--
```

9-1-2 Soundイベントを処理する

サウンドデータをロードする場合、イベントにより状況を確認する必要があります。通常はロードに失敗したり、読み込み中のオブジェクトを再生することができないためです。Soundオブジェクトは下表のイベントを送出します。再生を開始するタイミングは、completeイベントの発生後になります。

Soundクラスの主なイベント

タイミング	イベント名	イベントの概要
サウンドのロード開始	open (定数Event.OPEN)	サウンドのロード処理が開始する直前に送出されます。
サウンドのロードプロセス中	progress (定数ProgressEvent.PROGRESS)	サウンドのロードプロセスでデータがファイルまたはストリームから受信される間、定期的に送出されます。
ID3データが使用可能	id3 (定数Event.ID3)	MP3サウンドでID3データが使用可能になると送出されます。
サウンドのロード終了	complete (定数Event.COMPLETE)	サウンドデータのロードが完了すると送出されます。
サウンドのロード失敗	ioError (定数IOErrorEvent.IO_ERROR)	サウンドファイルが見つからない場合、またはサウンドデータのロードプロセスが中断されると送出されます。

Soundクラスのメソッド詳細

メソッド	メソッドの詳細
`load( stream:URLRequest, context: SoundLoaderContext = null ):void`	**stream**：ロードするMP3サウンドファイルを表すURLRequestオブジェクトです。このURLで指定される外部MP3ファイルのロードを開始します。 load()メソッドによるサウンドリソースのロード時にもFlash Playerのセキュリティサンドボックスは有効です。サンドボックス違反となるロードを実行した場合、「セキュリティ Sandbox違反」例外が発生します。セキュリティポリシーファイルを設置してクロスドメインアクセスを行う場合、checkPolicyFileプロパティをtrueに設定したSoundLoaderContentオブジェクトをcontextパラメータに指定します。
`close():void`	ストリーミングサウンドのロードと再生を中止します。ストリームされていないサウンドの場合、このメソッドによりエラーが発生します。

Soundクラスのメソッド詳細（続き）

メソッド	メソッドの詳細
play(startTime:Number = 0, loops:int = 0, sndTransform: SoundTransform = null):SoundChannel	サウンドを再生するSoundChannelオブジェクトを新しく作成して返します。このオブジェクトにより、サウンドの停止、ボリュームの監視を行います。ボリュームの制御、パン、バランスの設定を行うには、サウンドチャンネルに割り当てられたSoundTransformオブジェクトを使用します。 **startTime**：再生を開始する位置をミリ秒単位で設定できます。 **loop**：再生を繰り返す回数を指定できます。 **sndTransform**：ボリュームとパン情報を持つSoundTransformオブジェクトを割り当てることで、再生時のボリュームとパンを指定できます。

SAMPLE CODE

MP3サウンドをロードしてイベントを監視する

SoundSampleB.fla

```
import flash.events.Event;
import flash.events.ProgressEvent;
import flash.media.Sound;
import flash.net.URLRequest;

//Soundオブジェクト変数
var sound:Sound;
//URLRequestオブジェクト変数
var url:URLRequest;

//初期化処理関数
function init():void {
  //ロード開始ボタンのイベントハンドラ登録
  start_btn.addEventListener(MouseEvent.CLICK,startLoad);
}

//サウンドデータをロードする関数
function startLoad(event:MouseEvent):void {
  //Soundオブジェクトの生成とイベントリスナー登録
  sound = new Sound();
  sound.addEventListener(ProgressEvent.PROGRESS, soundLoadProgress);
  sound.addEventListener(Event.COMPLETE, soundLoadComplete);
  sound.addEventListener(IOErrorEvent.IO_ERROR, soundIOError);
  output( url_txt.text + " をロードします" );
  url=new URLRequest(url_txt.text);
```

```
  //ロードを開始
  sound.load( url );
}

//Progressイベントハンドラ ①
function soundLoadProgress(event:ProgressEvent):void {
  //ロード済みのサイズからパーセント算出
  var percentLoaded:uint =
  Math.round( (event.bytesLoaded / event.bytesTotal) * 100 );
  output("Progress イベント:" + percentLoaded + " %ロードしました" );
}

//ロード完了イベントハンドラ
function soundLoadComplete(event:Event):void {
  output("Complete イベント: ロードが完了しました" );
  //サウンドオブジェクトのplay()メソッドで再生を開始
  sound.play();
}                  sound.play();

//IOErrorイベントハンドラ
function soundIOError(event:IOErrorEvent) {
  output("例外が発生しました。" + e.name + ":" + e.message );
  output("IOError イベント: ロードが失敗しました" );
}

//メッセージを出力する関数
function output(msg:String):void {
  output_ta.text += msg+"¥n";
}

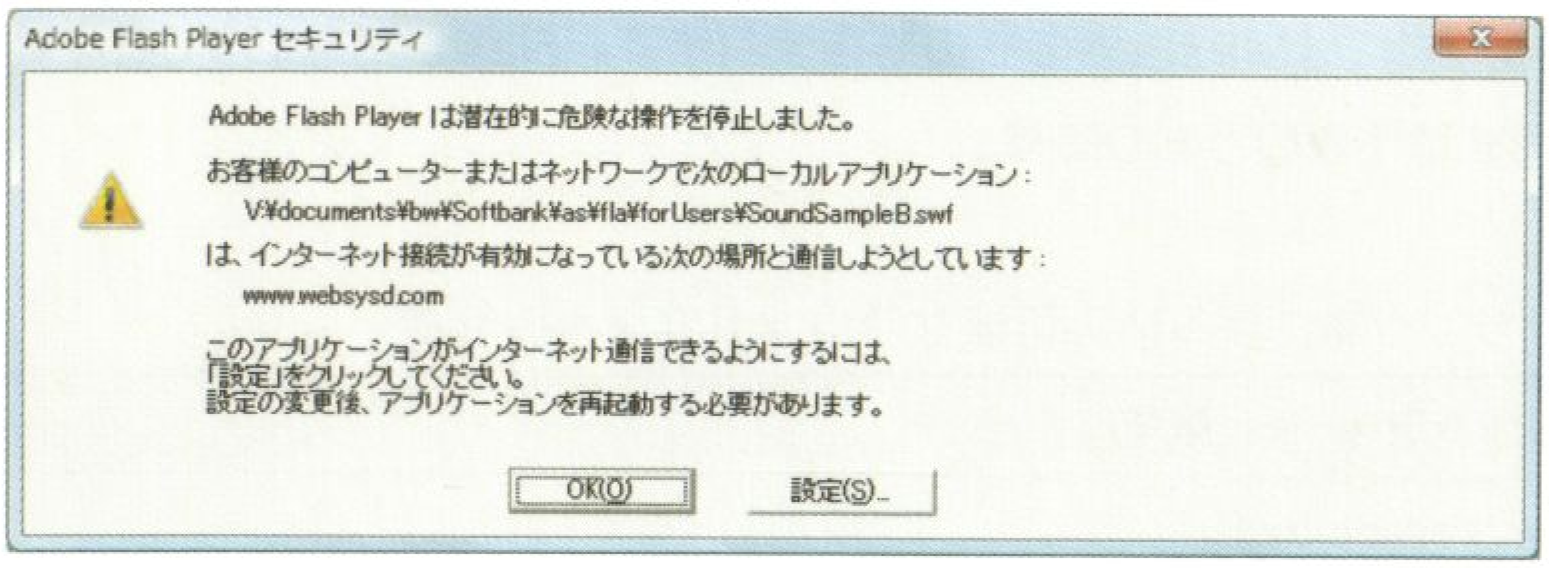
init();
```

MP3サウンドをロードする際に行う基本処理です。イメージやSWFのロード時と同様に、サウンドのロード時にもprogressイベントが発生してロード済みバイト数を取得できます。その処理は①のイベントハンドラで行っています。終了するとcompleteイベント、失敗するとioErrorイベントが発生します。ロード時に考慮すべきこととして、Flash Playerのセキュリティ違反があります。例えばローカルSWFを実行時にインターネット上のWebサーバからMP3をロードしようとすると、次のエラーとともに処理を終了します。

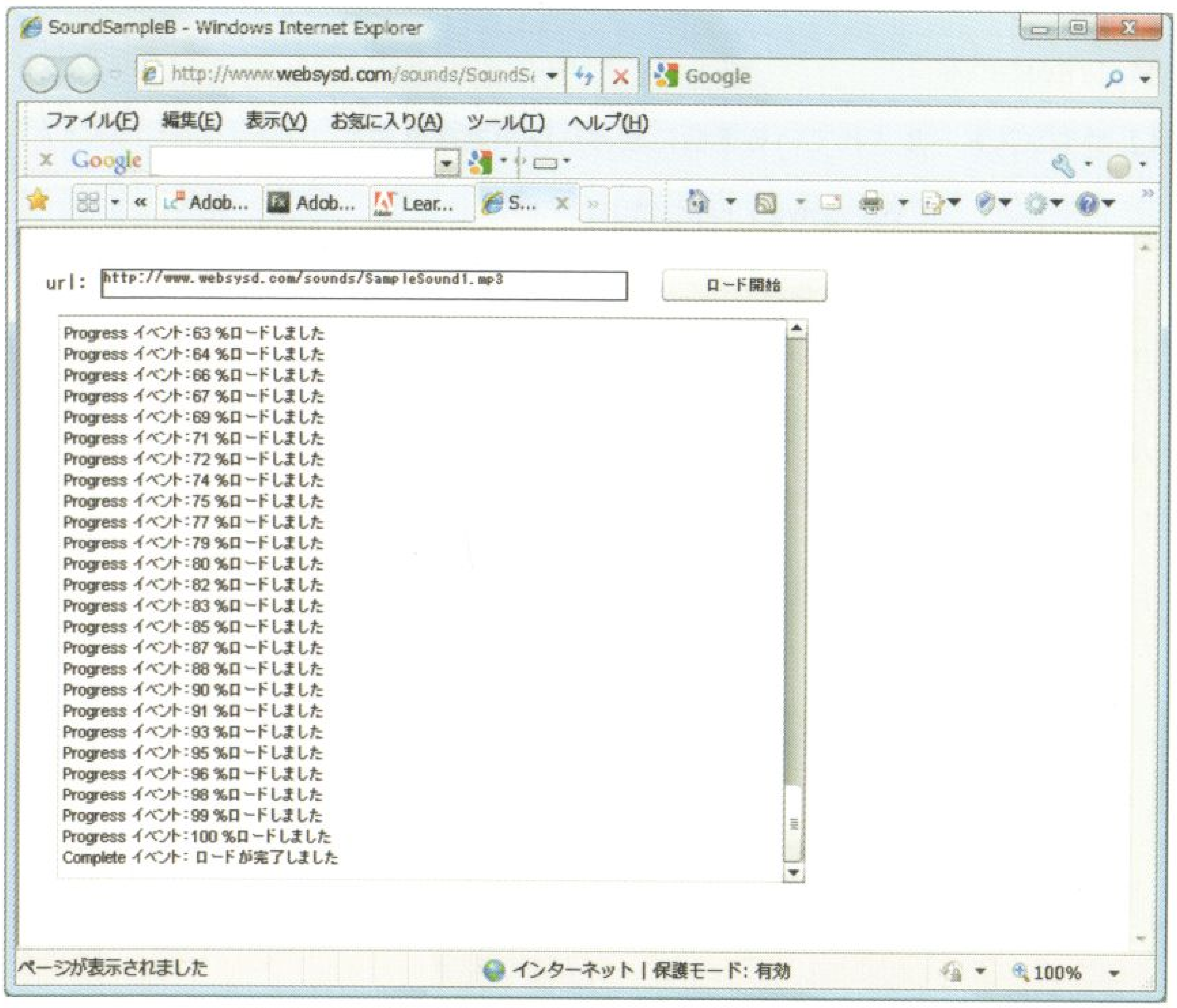
▶ 実行結果 出力ビューのエラー（一部）

```
*** セキュリティ Sandbox 違反 ***
http://www.websysd.com/sounds/SampleSound1.mp3 への接続は中止されました -
file:///C|/ SoundSampleB.swf で許可されません
SecurityError: Error #2147: URL
```

■ ブラウザで実行時のセキュリティエラーダイアログ



■ 正常ロード時



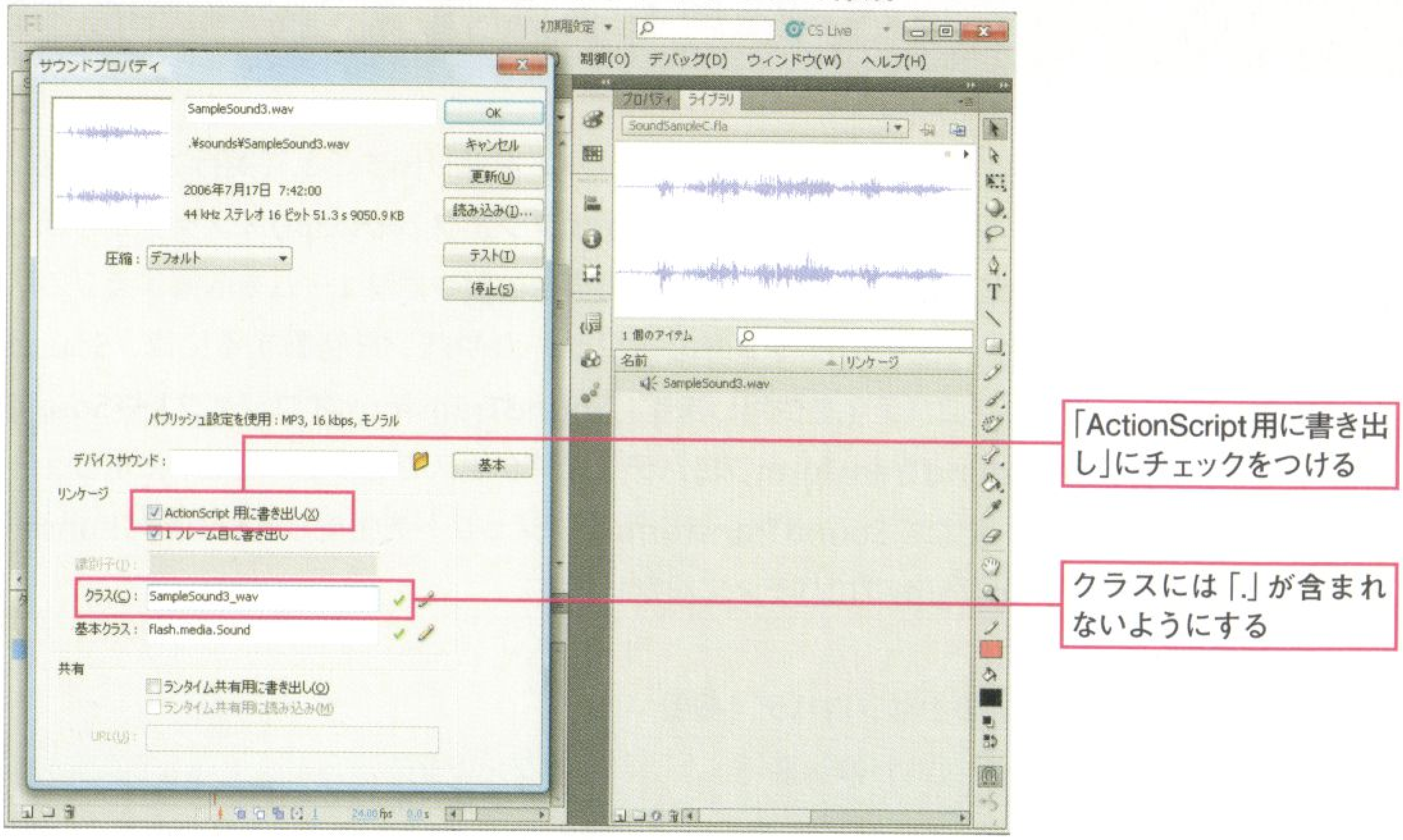
9-1-3 埋め込みサウンドを処理する

外部ファイルのロードではなく、サウンドをSWFに埋め込んで再生できます。サウンドを埋め込むには、オーサリング時にサウンドファイルをライブラリに読み込み、「詳細」プロパティで「ActionScript用に書き出し」チェックとともに、クラス名をつけておきます。クラス名には「.」を含めることができないため、「_」に置き換えるなどします。埋め込みサウンドの場合、利用可能なサウンドファイル形式にはMP3以外のものも含まれます。Flash CS4/CS5で読み込み可能なサウンドファイル形式は下表のとおりです。

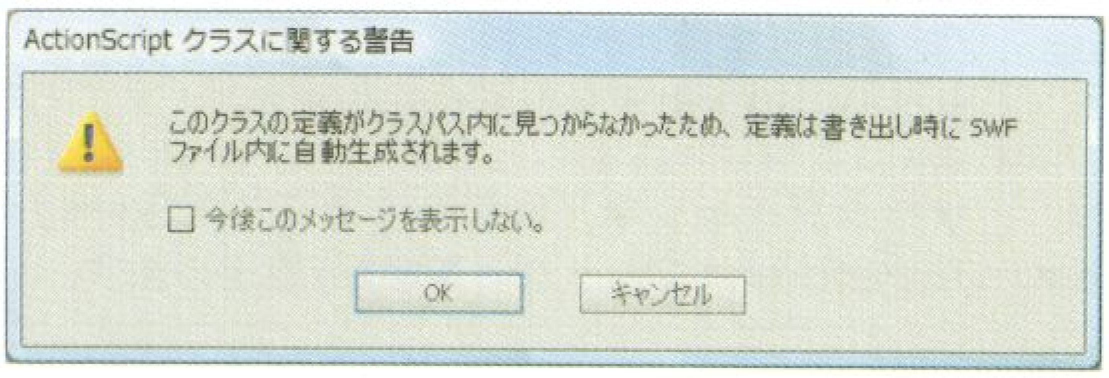
FlashCS4/CS5でオーサリング時に読み込み可能なサウンドファイル形式

ファイル形式	対応プラットフォーム（備考）
MP3	Windows / Macintosh
ASND	Windows / Macintosh（Adobe Soundboothのネイティブサウンド形式）
WAV	Windows のみ
AIFF	Macintoshのみ
以下はQuickTime4以上がインストールされている場合に読み込み可能な形式	
AIFF	Windows / Macintosh
Sound Designer II	Macintoshのみ
Sound Only QuickTime movies	Windows / Macintosh
Sun AU	Windows / Macintosh
System 7 Sounds	Macintoshのみ
WAV	Windows / Macintosh

■ サウンドプロパティによる埋め込みサウンドクラスの作成



■ サウンドクラスプロパティ設定時の警告



SAMPLE CODE

埋め込みサウンドの再生

SoundSampleC.fla

```
var s:SampleSound3_wav = new SampleSound3_wav();
s.play();
```

上記の「サウンドプロパティによる埋め込みサウンドクラスの作成」で使用したWAVファイルを使用します。ファイル名はSampleSound3.wavですが、このファイルを読み込んでライブラリプロパティの詳細で「ActionScript用に書き出し」にチェックをつけると、自動的に「SampleSound3.wav」というクラス名が生成されます。すでに解説したように、ここで「. 」は含めることができないため、「.wav」を取り除くか、「_ 」に置き換えるなどします。ここでは、「SampleSound3_wav」というクラス名にしました。OKを押してダイアログを閉じると「サウンドクラスプロパティ設定時の警告」が表示されますが、これはエラーではありません。クラス定義ができたらサンプルコードのようにインスタンスを生成してSoundクラスのplay()メソッドにより、wavサウンドが再生できます。

9-1-4 サウンドの再生と停止、ボリュームとパンを制御する

PART
1
2
3
4
5
6
7
8
9

サウンドの再生はSoundオブジェクトでplay()メソッドを呼び出すことで実行されますが、停止するには、play()メソッドが返すSoundChannelオブジェクトのstop()メソッドを呼び出す必要があります。SoundChannelクラスには現在の再生位置やボリュームを取得するプロパティが用意されています。再生中のサウンドのボリュームやパンを制御するには、Sound Transformオブジェクトのプロパティを使用します。SoundTransformオブジェクトはSound ChannelオブジェクトのsoundTransformプロパティでアクセスできます。再生中にボリュームやパンを変更するには、新たにSoundTransformオブジェクトを生成してSoundChannelオブジェクトのsoundTransformプロパティに設定します。

SoundChannelクラスの主なプロパティ概要

目的	プロパティ名	プロパティの概要
左/右チャンネルの現在のボリュームを取得する	leftPeak rightPeak :Number	[読み取り専用] 左/右チャンネルの現在のボリュームを取得します。0=最小(無音)、1=最大を表します。
再生ポジションを取得する	position:Number	[読み取り専用] サウンドファイル内で再生されている現在のポイントを示します。
SoundTransformオブジェクトを取得/設定する	soundTransform :SoundTransform	サウンドチャンネルに割り当てられたSound Transformオブジェクトを参照/設定します。

SoundChannelクラスの主なメソッド概要

目的	メソッド名	メソッドの概要
再生を停止する	stop():void	再生しているサウンドを停止します。

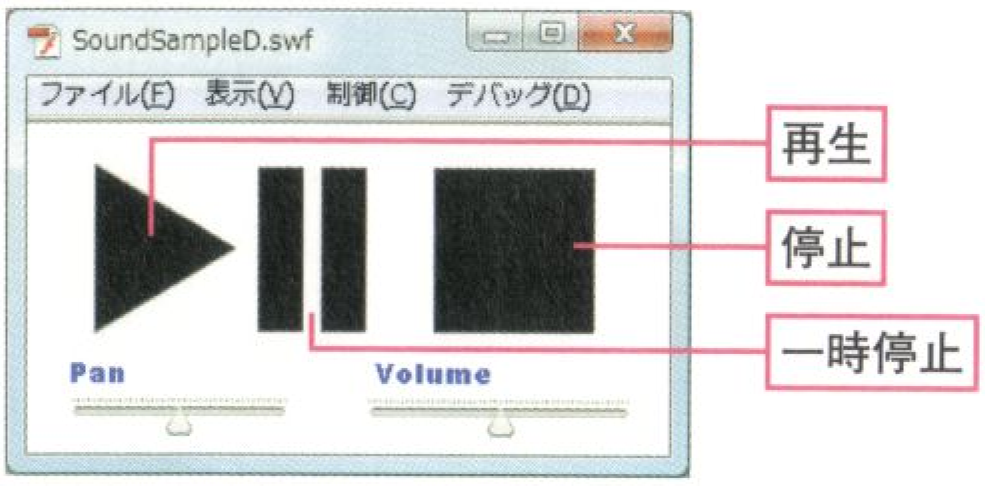
SundTransformクラスの主なプロパティ概要

目的	プロパティ名	プロパティの概要
ボリュームを変更/取得する	volume:Number	現在のボリュームを変更/取得します。0=最小(無音)、1=最大を表します。
左右のバランスを変更/取得する	pan:Number	左右のパンを設定/取得します。-1=左にパン、0=中央、1=右にパンを表します。

soundTransformクラスの主なメソッド概要

目的	メソッド名	メソッドの概要
コンストラクタ	SoundTransform(vol:Number = 1, panning:Number = 0)	コンストラクタによりvolume、panプロパティをあらかじめ設定したオブジェクトを作成し、SoundChannelオブジェクトのsoundTransformプロパティに適用することで再生中にボリュームとパンを変更することができます。Soundオブジェクトのplay()メソッドにSoundTransformオブジェクトを渡して再生開始時のボリュームとパンを決定することもできます。

パブリッシュ設定



SAMPLE CODE

サウンドの再生と停止、ボリュームとパン

SoundSampleD.fla

```
//mp3サウンドファイル
var snd:SampleSound3_wav;
//SoundChannelオブジェトド変数
var snd_cnl:SoundChannel;
//再生位置を保持する変数
var snd_pos:int = 0;
//SoundTransformオブジェクト変数
var st:SoundTransform;
//初期処理を行う関数
function init():void {
  //サウンドファイルのインスタンスを生成
  snd = new SampleSound3_wav();
  play_mc.addEventListener( MouseEvent.CLICK, playit );
  pause_mc.addEventListener( MouseEvent.CLICK, pauseit );
  stop_mc.addEventListener( MouseEvent.CLICK, stopit );
  //SoundTransformオブジェクトを生成
  st = new SoundTransform( 0.5, 0 );
  //Panのスライダーイベントリスナー
  pan_sldr.addEventListener( Event.CHANGE, setPan );
  //Volumeのスライダーイベントリスナー
  vol_sldr.addEventListener( Event.CHANGE, setVolume );
```

```
}
//再生を行う関数
function playit( event:Event ):void {
  snd_cnl = snd.play( snd_pos ); ――①positionプロパティの値をplay()メソッドの引数に渡す
  snd_cnl.soundTransform = st;
}
//一時停止を行う関数
function pauseit( event:Event ):void {
  if ( snd_cnl == null ) return;
  //再生位置を保持
  snd_pos = snd_cnl.position; ――②positionプロパティの値を保持
  snd_cnl.stop();
}
//停止を行う関数
function stopit( event:Event ):void {
  if ( snd_cnl == null ) return;
  //再生ポジションを先頭に
  snd_pos = 0;
  snd_cnl.stop();
}
//Panを変更する関数
function setPan( event:Event ):void {
  st.pan = (event.target as Slider).value; ――③Panの値をスライダーから取得
  snd_cnl.soundTransform = st; ――④Panの値をSoundChannelオブジェクトのsoundTransformプロパティに渡す
}
//Volumeを変更する関数
function setVolume( event:Event ):void {
  st.volume = (event.target as Slider).value; ――⑤ボリュームを取得
  snd_cnl.soundTransform = st; ――⑥ボリュームをSoundChannelオブジェクトのsoundTransformプロパティに渡す
}
init();
```

ライブラリにあるサウンドを再生するサンプルです。サウンドオブジェクトの作成は、9-1-3のサンプルと同様です。このサンプルでは、一時停止と停止、パンとボリュームの変更を行います。一時停止したサウンドを再開する場合、②で行っているように、positionプロパティの値を保持しておき、①のようにplay()メソッドの引数に渡す必要があります。Panで左右のバランスを変更する場合とボリュームを変更する場合、SoundTransformオブジェクトを使用する必要があります。

9-2-1 マイクを使用する

Microphoneクラスを使用して、マイクなどのサウンド入力デバイスの情報を取得したり、オーディオ入力をキャプチャーできます。ActionScriptでマイクを使用するには、まずMicrophoneクラスのstaticメソッドgetMicrophone()を呼び出して、インスタンスを取得します。その際、ブラウザで実行するFlashアプリケーションの場合、マイクへのアクセス許可を設定するプライバシーダイアログが表示され、ユーザが拒否すると、アプリケーションではStatusEvent.STATUSイベントが発生して、Microphoneオブジェクトのmutedプロパティがtrueになります。このプロパティは、ActionScriptでは変更できません。マイクからの入力をスピーカーに送るには、setLoopback()メソッドを呼び出します。音声入力はリソースを消費するため、マイクからの入力レベルが小さいか、一定期間入力がない場合、マイクがセッションを終了し、再び一定以上の入力があった場合にセッションを開始します。このタイミングを知らせるイベントが"activity"です。最低入力レベルと期間の設定は、setSilenceLevel()メソッドで行います。

Microphoneクラスの主なプロパティ

目的	プロパティ名	プロパティの概要
現在のサウンド量を調べる	activityLevel :Number	[読み取り専用] マイクが検知している音量を表します。
コーデック名を取得/設定する	codec:String	オーディオ圧縮に使用するコーデック名を取得/設定できます。
入力ゲインを取得/変更する	gain:Number	マイクによるサウンドのゲイン(増幅率)を取得/設定できます。
ユーザのマイクアクセス許可を調べる	muted:Boolean	[読み取り専用] ユーザがマイクへのアクセスを許可している場合、falseになります。拒否するとtrueになります。
現在のサウンドキャプチャデバイスの名前を取得する	name:String	[読み取り専用] サウンドキャプチャハードウェアから返される現在のサウンドキャプチャデバイスの名前を表します。
使用可能なすべてのサウンド入力デバイスの名前の配列を取得する	names:Array	[静的] [読み取り専用] 使用可能なすべてのサウンドキャプチャデバイス名のストリング配列です。
入力オーディオのサンプルレートを取得/変更する	rate:int	マイクのサウンドキャプチャレート(kHz)を表します。
マイクを有効にするために必要なサウンド量を指定する	silenceLevel :Number	[読み取り専用] マイクを有効化したり、activityイベントを送出するために必要な音量を表します。

Microphoneクラスの主なプロパティ（続き）

目的	プロパティ名	プロパティの概要
マイクが無音状態であると判断されるまでの時間を指定する	silenceTimeout : int	[読み取り専用] マイクがサウンド検知を停止してからactivityイベントが送出されるまでの時間（ミリ秒単位）を表します。

Microphoneクラスの主なメソッド概要

目的	メソッド名	メソッドの概要
Microphoneオブジェクトを取得する	getMicrophone(index:int = -1):Microphone	オーディオをキャプチャするMicrophoneオブジェクトを取得します。通常indexは渡しません。
ローカルスピーカーに音声を出力する	setLoopBack(stage:Boolean=true):void	マイクでキャプチャされたオーディオをローカルスピーカーに転送します。
サウンドと見なす最小入力レベルを設定する	setSilenceLevel(silenceLevel:Number, timeout:int = -1): void	そのレベル以上をサウンドと見なす最小入力レベルと停止を判定する時間を設定できます。 **silenceLevel**：0～100で最小入力レベルを設定します。100にするとマイクがサウンドを検知しません。 **timeout**：ミリ秒でサウンドが停止したと判定する時間を設定します。
オーディオフィードバックが発生するリスクは軽減	setUseEchoSuppression(useEchoSuppression: Boolean):void	オーディオコーデックのエコー抑制機能を使用するかどうかを指定できます。 **useEchoSuppression**：trueでエコー制御を行います。

Microphoneクラスの主なイベント

タイミング	イベント名	イベントの概要
セッションの開始/終了	activity (定数ActivityEvent.ACTIVITY)	マイクがセッションを開始または終了したときに送出されます。
マイクステータスの変更	status (定数StatusEvent.STATUS)	マイクがそのステータスをレポートしたときに送出されます。

SAMPLE CODE

Microphoneオブジェクトを使う

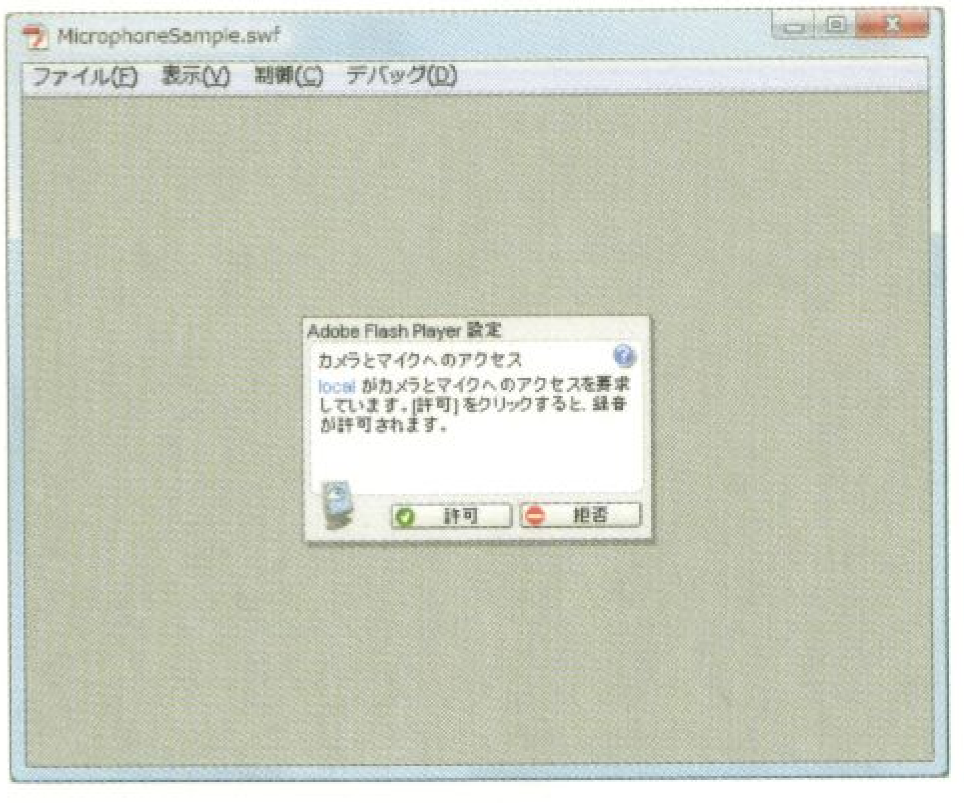
MicrophoneSample.fla

```
import flash.events.ActivityEvent;
import flash.events.StatusEvent;
import flash.media.Microphone;
//Microphoneオブジェクト変数
var mic:Microphone;
var devices:Array;
```

```
function init():void {
  //Microphoneオブジェクトの取得
  mic=Microphone.getMicrophone(); ―――――――――――――― ①
  if ( mic == null ) {
    trace( "接続された入力デバイスはありません。" );
    return; ―――――――――――――――――――――――――――― ②
  }
  //デバイス名の取得表示
  devices=Microphone.names
  trace( "入力デバイスの数:" + devices.length );
  for (var i:int = 0; i < devices.length; i++) {
    trace("入力デバイス(" + i + "): " + devices[i]);
  }
  //Microphoneオブジェクトのプロパティ
  //ゲイン(増幅率)
  mic.gain = 40;
  //キャプチャーレート
  mic.rate = 11;
  //エコー制御を行う
  mic.setUseEchoSuppression(true);
  //ローカルスピーカーに転送
  mic.setLoopBack(true);
  //最小入力レベルとタイムアウト時間
  mic.setSilenceLevel(20, 3000); ―――――――――――――― ③
  //Microphoneオブジェクトのイベントリスナー
  mic.addEventListener(ActivityEvent.ACTIVITY, micAct );
  mic.addEventListener(StatusEvent.STATUS, micStatus );
}
function micAct(event:ActivityEvent):void {
  trace("activity イベント: " +
    "activating(デバイス=active)=" + event.activating +
    ", Microphone..activityLevel=" + mic.activityLevel);
}
function micStatus(event:StatusEvent):void {
  trace("status イベント: " +
    " level(カテゴリ)=" + event.level +
    ", code(ステータス)=" + event.code +
    ", Microphone.muted=" + mic.muted ); ―――――――――― ④
}
init();
```

2
1

①でMicrophoneオブジェクトを取得していますが、この時点で接続されたマイクがない場合、micはnullとなるので、②でreturnして終了します。マイクが接続されている場合、下記のようなFlash Player設定ダイアログが表示されます。ユーザが拒否すると、statusイベントが発生してイベントハンドラが呼び出されます。その場合、④のmutedプロパティがtrueになります。ユーザが許可すると、③で設定したマイクの最低入力レベルと時間に基づいてマイクのセッションが切断/アクティブと変化します。そのたびにstatusイベントが発生します。



▶ 実行結果 ※ユーザが拒否した場合の出力例

```
入力デバイスの数:1
入力デバイス(0): マイク (2- Realtek High Definit
status イベント: level(カテゴリ)=status, code(ステータス)=Microphone.Muted,
Microphone.muted=true
```

▶ 実行結果 ※ユーザが許可した場合の出力例

```
入力デバイスの数:1
入力デバイス(0): マイク (2- Realtek High Definit
status イベント: level(カテゴリ)=status, code(ステータス)=Microphone.
Unmuted, Microphone.muted=false
activity イベント: activating(デバイス=active)=true, Microphone..
activityLevel=25
activity イベント: activating(デバイス=active)=false, Microphone..
activityLevel=0
activity イベント: activating(デバイス=active)=true, Microphone..
activityLevel=44
:
```

9－3 ビデオ

9-3-1 FLVPlayback コンポーネントでビデオを再生する

FLV（Flash Video）形式はWebではもっとも使用されているビデオフォーマットです。Flashはその配信に関して強力なソリューションを提供します。Flashでビデオを配信する場合、次のようないくつかの方式が選択可能です。

- **Adobe Flash Media Server によるビデオのストリーミング**
- **Web サーバーからのビデオのプログレッシブダウンロード**
- **Flash ドキュメントへのビデオの埋め込み**

クライアントをFlashで構築する場合、FLVPlaybackコンポーネントが用意されており、簡単な設定でストリーミングおよびプログレッシブダウンロードによる再生を行うことができます。FLVPlaybackには再生やボリュームコントロール、巻き戻しボタンなどを備えたスキンが用意されており、選択できます。プロパティの設定のみで再生を行うことができますが、メソッドも多く用意されており、ActionScriptによる処理を追加できます。FLVPlaybackによるビデオの再生手順は次のようになります。

■ FLVPlayback クラスでビデオ再生を行う手順

No.	作業内容
1	FLVPlaybackコンポーネントをステージに配置してプロパティを設定します。 ビデオソースを表すプロパティはActionScript 3の場合、sourceで、ここにビデオコンテンツまたはXMLファイルのURLを設定します。設定はデザイン時にも行えます。
2	デザイン時にsourceを指定している場合、Flashムービーを再生するとビデオがロードされ、完了後再生が始まります。ActionScriptでロードを開始する場合、load()メソッドを使用します。 ActionScriptでイベントを取得して状況を把握できます。

■ FLVPlayback コンポーネント



FLVPlaybackクラスの主なプロパティ概要

目的	プロパティ名	プロパティの概要
自動再生を行う	autoPlay :Boolean	trueの場合、sourceプロパティの設定後に自動的にFLVファイルが再生されます。
自動巻き戻しを行う	autoRewind :Boolean	trueの場合は、プレーヤーでストリームの最後に到達するか、stop()メソッドが呼び出されて再生が停止したときに、FLVファイルをフレーム1に巻き戻します。
ロード済みバイト数を取得する	bytesLoaded :uint	[読み取り専用] HTTPダウンロードの進捗状況を示す数値(バイト数)です。
総バイト数を取得する	bytesTotal:uint	[読み取り専用] HTTPダウンロードでダウンロードされた総バイト数を指定する数値です。
FLVPlaybackの幅と高さ	height:Number width:Number	FLVPlayback インスタンスの高さ/幅を指定します。
ストリーミングライブ指示を行う	isLive:Boolean	ビデオストリームがライブである場合にtrueになります。
再生済みのパーセントを取得する	playheadPercentage:Number	現在のplayheadTime位置のtotalTimeにおけるパーセンテージを表します。
現在の再生位置を調べる	playheadTime :Number	現在の再生ヘッドの時間または位置を秒単位で表します。
ソースFLVファイルのサイズを調べる	preferredHeight :int	[読み取り専用] ソースFLVファイルの高さ/幅を指定する数値です。
ロードしたビデオのサイズを決める	scaleMode :String	ロード後のビデオサイズをFLVサイズにするか、FLVの再生サイズをビデオサイズに合わせるかを指定します。
FLVファイルのURLを指定/取得する	source:String	ストリーミング/ダウンロードするFLVファイルのURLとストリーミング方法を指定するストリングです。
ビデオ再生時間の合計を取得する	totalTime :Number	ビデオ再生時間の合計を表す数値(秒単位)です。
ボリュームコントロール設定値を設定/取得する	volume:Number	ボリュームコントロール設定値を示す0〜1の間の数値です。

FLVPlaybackクラスの主なメソッド概要

目的	メソッド名	メソッドの概要
フルスクリーン表示する	enterFullScreenDisplayState():void	FLVPlaybackビデオプレーヤーをフルスクリーンにします。
FLVファイルのロードを開始する	load(source:String, totalTime:Number = NaN, isLive:Boolean = false):void	sourceで指定したFLVファイルのロードを開始します。 **totalTime**:オプションでビデオの再生時間を示します。 **isLive**:ライブストリームの場合、trueにします。

FLVPlaybackクラスの主なメソッド概要(続き)

目的	メソッド名	メソッドの概要
再生を一時停止する	`pause():void`	ビデオストリームの再生を一時停止します。
再生を開始する	`play(` `source:String = null,` `totalTime:Number = NaN,` `isLive:Boolean = false` `):void`	ビデオストリームを再生します。
シークする	`seek(time:Number):void`	ファイル内の指定時間(秒単位)を小数点以下3桁までの精度(ミリ秒精度)でシークします。
再生を停止する	`stop():void`	ビデオの再生を停止します。

FLVPlaybackクラスの主なイベント

タイミング	イベント名	イベントの概要
再生が完了した	`complete` (定数VideoEvent.COMPLETE)	プレーヤーがFLVファイルの最後に到達して再生が完了したときに送出されます。
ダウンロード状況が更新された	`progress` (定数VideoProgressEvent.PROGRESS)	進捗状況はダウンロードしたバイト数で表されます。
FLVファイルがロードされ表示準備ができた	`ready` (定数VideoEvent.READY)	FLVファイルがロードされ表示準備ができたときに送出されます。
再生ヘッドの位置が変更された	`seeked` (定数VideoEvent.SEEKED)	seek()の呼び出し、playheadTimeプロパティの設定、またはSeekBarコントロールの使用により、再生ヘッドの位置が変更されたときに送出されます。

SAMPLE



PART 1 2 3 4 5 6 7 8 9

SAMPLE CODE

FLVPlaybackを使う

VideoSampleA.fla

```
import fl.video.VideoProgressEvent;
import fl.video.VideoEvent;

var videoURL:String = "videos/sampleFLV.flv";

function init():void {
  flvPlayback.addEventListener( VideoProgressEvent.PROGRESS, video
  Progress );
  flvPlayback.addEventListener( VideoEvent.READY, videoReady );
  flvPlayback.addEventListener( VideoEvent.SEEKED, videoSeeked );
  flvPlayback.load(videoURL); ―――――――――――――――――――――――――――――――― ①
}
function videoProgress(event:VideoProgressEvent ):void {
  trace( event.bytesTotal + " 中 " +  event.bytesLoaded + " バイトロード完了"
  ); ―――――――――――――――――――――――――――――――――――――――――――――――――― ②
}
function  videoReady(event:VideoEvent ):void {     ┐
  trace( "再生の準備ができました。ソースビデオサイズ:縦=" +  │
    flvPlayback.preferredHeight +                  ├―――――――― ③
    ", 横=" + flvPlayback.preferredWidth );         │
}                                                  ┘
function videoSeeked(event:VideoEvent ):void {     ┐
  trace( "再生ヘッド位置が変わりました。state=" + event.state ); ├―― ④
}                                                  ┘
init();
```

FLVPlaybackコンポーネントを使用すると、単にビデオを再生する場合はほとんどプログラミングなしに行うことができます。デザインモードでコンポーネントのsourceを指定していない場合、または動的に再生するビデオを変える場合は、①で行っているように、load()メソッドでファイル名を指定します。サーバーからロードされる状況を取得するには、progressイベントを使用します。②のように、ロード済みバイト数を取得できます。ロードが完了し、再生の準備が整うと、readyイベントが発生して、③のように処理できます。また、ビデオの再生位置が変化した場合、seekedイベントが発生して、④のように情報を得ることができます。

9-3-2 ActionScriptでリモートビデオ再生を行う

ActionScriptでビデオのロード、表示、再生のすべてを制御することができます。外部ビデオをロードする場合、Web サーバーからファイルをロードしてプログレッシブダウンロードを行うか、Flash Media Serverなどの専用サーバーから配信されるストリーミングビデオを再生することができます。ActionScriptでそのようなビデオソースを操作する場合、NetConnectionクラス、NetStreamクラス、Videoクラスを使用します。NetConnectionクラスはサーバ側アプリケーションインスタンスとの接続を行い、NetStreamクラスがストリーミングを行い、再生や停止の制御を行います。このとき、1つの接続に複数のストリームを割り当てることができます。Videoクラスは表示オブジェクトのサブクラスで、ステージ上で再生するビデオの表示を行います。ロードするビデオの情報は、NetStreamクラスのclientプロパティに指定するオブジェクトに、コールバック関数を設定するという方法で取得できます。コールバック関数の種類については、下表「NetStreamクラスのclientオブジェクトのプロパティ（コールバック関数用）」を、使用方法については、サンプルコードをご覧ください。サンプルコードでは、プログレッシブダウンロードによるビデオの再生を行います。

■ ビデオのロードと再生を制御するためのActionScriptクラス

クラス	概要
flash.net. NetConnection	ストリーミングサーバやWebサーバとクライアント間のデータストリームを構築するベースとなる接続を提供します。
flash.net. NetStream	NetConnectionオブジェクトにより接続されたストリーミングサーバとのコネクション上のチャネルとして、ストリーミング接続を開き、メディアの再生を制御します。
flash.media. Video	Videoクラスのインスタンスはステージ上のビデオコンテンツボックスとしてライブビデオやビデオファイルを再生表示します。VideoクラスはDisplayObjectのサブクラスであるため、配置や変換、フィルターの適用、ブレンドモードなどを使用して操作できます。

■ NetConnectionクラスの主なプロパティ概要

目的	プロパティ名	プロパティの概要
サーバーへの接続方法を調べる	`connected :Boolean`	[読み取り専用] 永続的なRTMP接続を介して(true)、またはそれ以外の方法で(false)サーバに接続されていることを示します。
NetConnectionインスタンスのAMFバージョンを設定する	`protocol:String`	このNetConnectionインスタンスのオブジェクトエンコード(AMFバージョン)を表します
接続プロトコルを調べる	`protocol:String`	[読み取り専用] Flash Media Serverとの接続プロトコルを表します。
接続失敗時の代替手段を指定する	`proxyType :String`	サーバへの初期接続に失敗した場合に実行する代替手段を指定します。

NetConnectionクラスの主なメソッド概要

目的	メソッド名	メソッドの概要
アプリケーションサーバ上のメソッドを呼び出す	`call( command:String, responder: Responder, ... arguments ):void`	Flash Media Server上またはFlash Remoting実行しているアプリケーションサーバ上のコマンドまたはメソッドを呼び出します。
実行されているすべてのNet Streamオブジェクトを切断する	`close():void`	ローカルで開始した接続またはサーバへの接続を閉じ、NetConnection.Connect.Closedをcodeプロパティとして、netStatusイベントを送出します。
サーバと接続する	`connect( command:String, ... arguments ):void`	Flash PlayerまたはAIRアプリケーションとFlash Media Serverアプリケーション、またはWebサーバとの間に接続を作成します。

NetConnectionクラスの主なイベント

タイミング	イベント名	イベントの概要
ステータスまたはエラーが送出された	netStatus (定数NetStatusEvent. NET_STATUS)	NetConnectionオブジェクトが、そのステータスまたはエラーの状況をレポートするときに送出されます。
セキュリティエラーが発生した	securityError (定数SecurityErrorEvent. SECURITY_ERROR)	NetConnection.call()の呼び出しによって呼び出し元のセキュリティサンドボックスの外部にあるサーバに接続しようとすると送出されます。

NetStreamクラスの主なプロパティ概要

目的	プロパティ名	プロパティの概要
コールバックメソッドが呼び出されるオブジェクトを指定する	`client :Object`	FLVのストリーミングやプログレッシブダウンロードの処理にコールバックメソッドが呼び出されるオブジェクトを指定します。

NetStreamクラスの主なメソッド概要

目的	メソッド名	メソッドの概要
マイクオーディオ入力をFlash Media Serverに送信する	`attachAudio( microphone:Microphone ):void`	microphoneで指定されるMicrophone オブジェクトからNetStreamオブジェクトを介して送信されるオーディオストリームを指定します。
カメラキャプチャー入力をFlash Media Serverに送信する	`attachCamera( theCamera:Camera, snapshotMilliseconds: int = -1):void`	カメラからビデオのキャプチャを開始します。theCameraがnullに設定されると、キャプチャを停止します。

NetStreamクラスの主なメソッド概要（続き）

目的	メソッド名	メソッドの概要
ストリーム再生を停止して開放する	`close():void`	ストリーム上のすべてのデータの再生を停止し、timeプロパティを0に設定し、他の用途でストリームにアクセスできるようにします。
ストリーム再生を一時停止する	`pause():void`	ビデオストリームの再生を一時停止します。
メディアを再生する	`play(... arguments):void`	FLV、F4V、MP3などのメディアファイルを再生します。
ストリーミングオーディオ、ビデオ、テキストをFlash Media Serverに送信する	`publish(name:String = null, type:String = null):void`	クライアントからFlash Media Serverへ、ストリーミングオーディオ、ビデオ、およびテキストメッセージを送信します。
ストリーム再生を再開する	`resume():void`	再生を一時停止していたビデオストリームを再開します。
ストリームをシークする	`seek(offset:Number):void`	offsetで指定された位置にもっとも近いキーフレームをシークします。

NetStreamクラスの主なイベント

タイミング	イベント名	イベントの概要
ネットワーク処理が失敗したとき	ioError（定数IOErrorEvent.IO_ERROR）	入出力エラーが発生してネットワーク処理が失敗したときに送出されます。
ステータスまたはエラーの状態をレポートしたとき	netStatus（定数NetStatusEvent.NET_STATUS）	NetStreamオブジェクトが、そのステータスまたはエラーの状態をレポートすると送出されます。

NetStreamクラスのclientオブジェクトのプロパティ（コールバック関数用）

タイミング	プロパティ名	プロパティの概要
ビデオに埋め込まれた記述的情報を受け取ったとき	onMetaData	再生中のビデオに埋め込まれた記述的情報をFlash Playerが受け取ったときに応答するリスナーを定義します。
ストリームの再生が終了したとき	onPlayStatus	NetStreamオブジェクトによるストリームの再生が終了した場合に応答するリスナーを定義します。
メディアファイルに埋め込まれたテキストデータを受信したとき	onTextData	Flash Playerが再生中のメディアファイルに埋め込まれたテキストデータを受信したときに応答するリスナーを定義します。

SAMPLE



SAMPLE CODE

プログレッシブダウンロードによるビデオ再生を行う VideoSampleB.fla

```
import flash.display.Sprite;
import flash.events.NetStatusEvent;
import flash.events.SecurityErrorEvent;
import flash.media.Video;
import flash.net.NetConnection;
import flash.net.NetStream;
import flash.events.Event;

//ロードするビデオファイル
var videoURL:String="videos/sampleFLV.flv";
var connection:NetConnection;
var stream:NetStream;

//NetConnection接続を行う処理
function init() { ──────────────────────────── ①
  connection = new NetConnection();
  //ステータス、エラー時のイベントハンドラ登録
  connection.addEventListener(NetStatusEvent.NET_STATUS, netStatus);
  //セキュリティエラー時のイベントハンドラ登録
  connection.addEventListener(SecurityErrorEvent.SECURITY_ERROR,
  netConnectionSecurityError);
  //サーバーと接続
  connection.connect(null);
}
```

```
//セキュリティエラー時のイベントハンドラ
function netConnectionSecurityError(event:SecurityErrorEvent):void {
  trace("セキュリティエラーが発生しました。: " + event.toString() );
}

//ステータス、エラー時のイベントハンドラ(NetConnection,NetStream共通)
function netStatus(event:NetStatusEvent):void { ——②
  //ステータスの内容を判定
  trace(event.info.code);
  switch (event.info.code) {
      //接続成功
    case "NetConnection.Connect.Success" :
      trace( "NetConnection接続に成功しました。" );
      getStream();
      break;
      //接続失敗
    case "NetStream.Play.StreamNotFound" :
      trace("NetStream接続に失敗しました。 " + videoURL);
      break;
      //その他のステータス/エラー
    default :
      switch (event.info.level ) {
        case "status" :
          trace("その他のステータスイベントが発生しました。code=" +
          event.info.code );
          break;
        case "error" :
          trace("その他のエラーが発生しました。code=" + event.info.code );
          break;
      }
      break;
  }
}

//NetStreamの処理
function getStream():void { ——③
  //NetStreamコンストラクタにNetConnectionオブジェクトを渡す
  stream=new NetStream(connection);
  //ステータス、エラー時のイベントハンドラ登録
  stream.addEventListener(NetStatusEvent.NET_STATUS, netStatus);
```

```
  //コールバックメソッド用オブジェクト
  var client:Object = new Object();
  client.onMetaData = this.onMetaData;
  client.onCuePoint = this.onCuePoint;
  stream.client = client;
  //表示用のVideoオブジェクト
  var video:Video = new Video();
  //NetStreamと関連づける
  video.attachNetStream(stream);
  stream.play(videoURL);
  addChild(video);
  //一時停止、再生、停止を行う関数の登録
  resume_mc.addEventListener(MouseEvent.CLICK, resumeIt);
  pause_mc.addEventListener(MouseEvent.CLICK, pauseIt);
  stop_mc.addEventListener(MouseEvent.CLICK, stopIt);
}

//コールバックメソッド
function onMetaData(info:Object):void {
  trace("コールバックメソッド(metadata): duration=" + info.duration + "
  width=" + info.width + " height=" + info.height + " framerate=" +
  info.framerate);
}
function onCuePoint(info:Object):void {
  trace("コールバックメソッド(cuepoint): time=" + info.time + " name=" +
  info.name + " type=" + info.type);
}

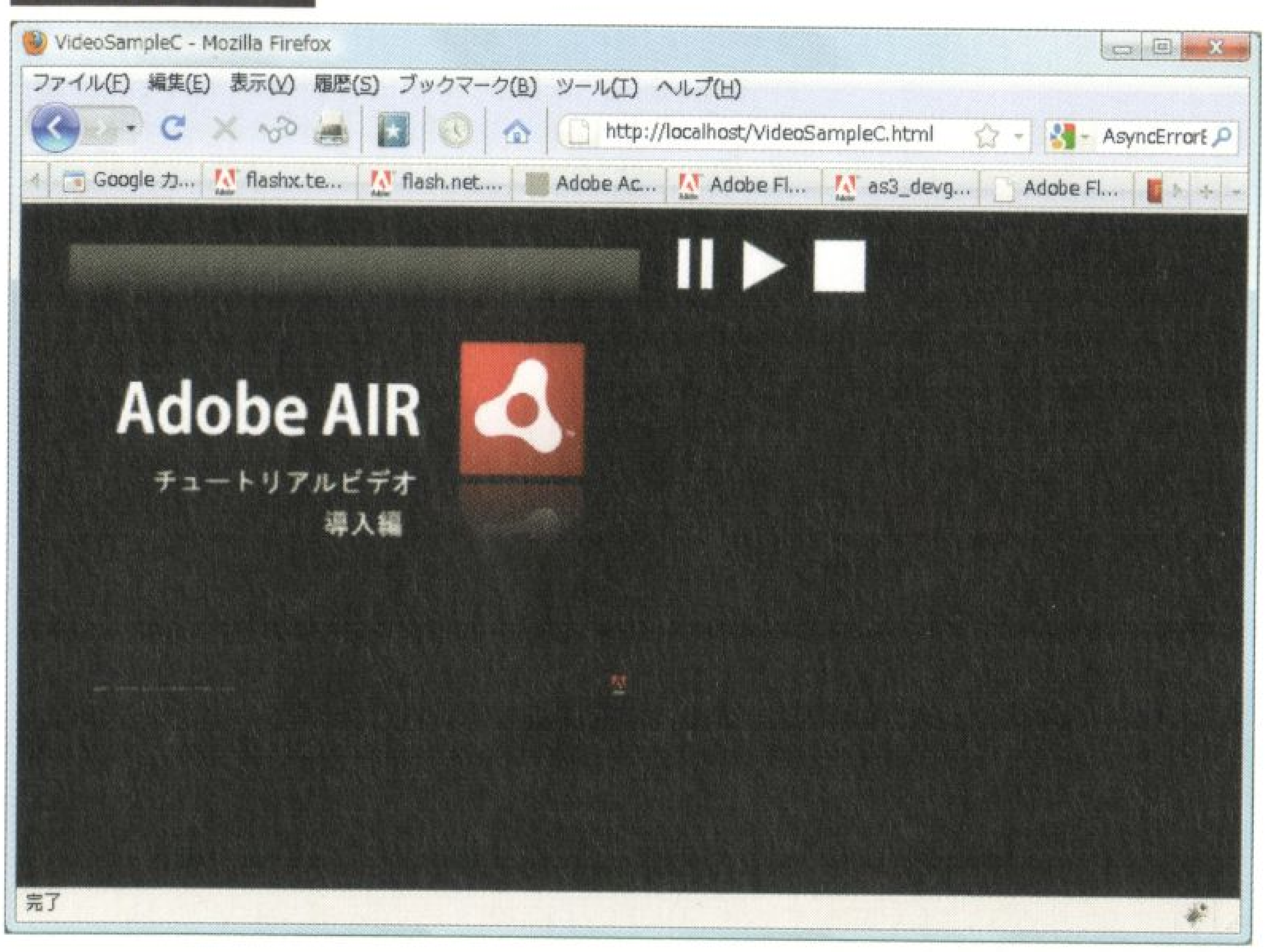
//一時停止、再生、停止を行う処理関数
function resumeIt(event:Event):void {
  stream.resume();
}
function pauseIt(event:Event):void {
  stream.pause();
}                                                    ④
function stopIt(event:Event):void {
  stream.pause();
  stream.seek( 0 );
}
init();
```

①からのinit()関数でNetConnection接続を行います。②からのステータスイベントハンドラで接続成功、エラーを判定できます。このハンドラはNetStreamのステータスハンドラと共通です。NetConnectionが成功すると、③からのgetStream()関数により、NetStreamの処理を行います。ここでは、metadata、cuepointに対するコールバック関数オブジェクトの設定を行い、表示用のVideoオブジェクトを作成したら、play()メソッドを呼び出して再生を行います。再生のコントロールは④で行っているように、NetStreamオブジェクトのメソッドで行います。

▶ 実行結果（出力例）

```
NetConnection.Connect.Success
NetConnection接続に成功しました。
NetStream.Play.Start
その他のステータスイベントが発生しました。code=NetStream.Play.Start
コールバックメソッド(metadata): duration=100.84426303854875 width=655
height=480 framerate=undefined
コールバックメソッド(metadata): duration=100.8 width=655 height=480 framerat
e=29.970016479492188
NetStream.Buffer.Empty
その他のステータスイベントが発生しました。code=NetStream.Buffer.Empty
NetStream.Buffer.Full
その他のステータスイベントが発生しました。code=NetStream.Buffer.Full
⋮
```

SAMPLE



SAMPLE CODE

ストリーミングによるビデオ再生（FMS3で確認）

VideoSampleC.fla

```
import flash.display.Sprite;
import flash.events.NetStatusEvent;
import flash.events.SecurityErrorEvent;
import flash.media.Video;
import flash.net.NetConnection;
import flash.net.NetStream;
import flash.events.Event;

//ストリーミングサーバURLとロードするビデオファイル名
var serverURL:String="rtmp://localhost/vod";
var videoFile:String="sampleFLV";
var connection:NetConnection;
var stream:NetStream;

//NetConnection接続を行う処理
function init() {
  connection = new NetConnection();
  //ステータス、エラー時のイベントハンドラ登録
  connection.addEventListener(NetStatusEvent.NET_STATUS, netStatus);
  //セキュリティエラー時のイベントハンドラ登録
  connection.addEventListener(SecurityErrorEvent.SECURITY_ERROR, netConnec
  tionSecurityError);
  //その他のエラー時のイベントハンドラ登録
  connection.addEventListener(AsyncErrorEvent.ASYNC_ERROR,
  netAnotherError);
  connection.addEventListener(IOErrorEvent.IO_ERROR, netAnotherError);
  //コールバックメソッド用オブジェクト
  var client:Object = new Object();
  client.onBWDone = this.onBWDone;
  client.onBWCheck = this.onBWcheck;
  connection.client = client;
  //サーバーと接続
  connection.connect(serverURL);
}

//セキュリティエラー時のイベントハンドラ
function netConnectionSecurityError(event:SecurityErrorEvent):void {
```

```
  trace("セキュリティエラーが発生しました。: " + event.toString() );
}

//その他のエラーハンドラ
function netAnotherError(event:Event ):void {
  trace("その他のエラーが発生しました。: " + event.toString() );
}

//ステータス、エラー時のイベントハンドラ(NetConnection,NetStream共通)
function netStatus(event:NetStatusEvent):void {
  //ステータスの内容を判定
  trace(event.info.code);
  switch (event.info.code) {
      //接続成功
    case "NetConnection.Connect.Success" :
      trace( "NetConnection接続に成功しました。" );
      connection.call("checkBandwidth", null);
      getStream();
      break;
      //接続失敗
    case "NetStream.Play.StreamNotFound" :
      trace("NetStream接続に失敗しました。 " + serverURL);
      break;
      //その他のステータス/エラー
    default :
      switch (event.info.level ) {
        case "status" :
    trace("その他のステータスイベントが発生しました。code=" + event.info.code );
          break;
        case "error" :
          trace("その他のエラーが発生しました。code=" + event.info.code );
          break;
      .}
      break;
  }
}

//NetStreamの処理
function getStream():void {
```

```
  //NetStreamコンストラクタにNetConnectionオブジェクトを渡す
  stream=new NetStream(connection);
  //ステータス、エラー時のイベントハンドラ登録
  stream.addEventListener( NetStatusEvent.NET_STATUS, netStatus);
  stream.addEventListener( AsyncErrorEvent.ASYNC_ERROR, netAnotherError );
  stream.addEventListener( IOErrorEvent.IO_ERROR, netAnotherError );
  //コールバックメソッド用オブジェクト
  var client:Object = new Object();
  client.onMetaData = this.onMetaData;
  client.onCuePoint = this.onCuePoint;
  stream.client = client;
  //表示用のVideoオブジェクト
  var video:Video = new Video();
  video.width = 400;
  video.height = 300;
  video.x = 10;
  video.y = 10;
  //NetStreamと関連付ける
  video.attachNetStream(stream);
  stream.play(videoFile);
  addChild(video);
  //一時停止、再生、停止を行う関数の登録
  resume_mc.addEventListener(MouseEvent.CLICK, resumeIt);
  pause_mc.addEventListener(MouseEvent.CLICK, pauseIt);
  stop_mc.addEventListener(MouseEvent.CLICK, stopIt);
}

//コールバックメソッド
function onMetaData(info:Object):void {
  trace("コールバックメソッド(metadata): duration=" + info.duration + " width=" +
  info.width + " height=" + info.height + " framerate=" + info.framerate);
}
function onCuePoint(info:Object):void {
  trace("コールバックメソッド(cuepoint): time=" + info.time + " name=" + info.
  name + " type=" + info.type);
}
function onBWDone(...rest):void {
  var bw:Number;
  if (rest.length>0) {
```

```
        bw=rest[0];
        trace("コールバックメソッド(bandwidth) bandwidth= " + bw + " Kbps.");
    }
}
function onBWCheck(... rest):Number {
    return 0;
}

//一時停止、再生、停止を行う処理関数
function resumeIt(event:Event):void {
    stream.resume();
}
function pauseIt(event:Event):void {
    stream.pause();
}
function stopIt(event:Event):void {
    stream.pause();
    stream.seek( 0 );
}

init();
```

▶ 実行結果（出力例）

```
NetConnection.Connect.Success
NetConnection接続に成功しました。
NetStream.Play.Reset
その他のステータスイベントが発生しました。code=NetStream.Play.Reset
NetStream.Play.Start
その他のステータスイベントが発生しました。code=NetStream.Play.Start
NetStream.Buffer.Full
その他のステータスイベントが発生しました。code=NetStream.Buffer.Full
コールバックメソッド(metadata): duration=37.637 width=656 height=480 framera
te=29.970016479492188
NetStream.Pause.Notify
その他のステータスイベントが発生しました。code=NetStream.Pause.Notify
NetStream.Buffer.Flush
⋮
```

3
2



クラス別INDEX

INDEX

INDEX

INDEX



INDEX

INDEX

S～U

V～X

あ行

か行

ま行

ら行

■著者プロフィール

宮田 亮（みやた りょう）

20年を超えるIT開発・教育・マネージメントキャリアを持つ。新卒で入社した通信メーカー系のシステム開発部に従事し、パケット交換機の基礎開発を行う。1990年代に独立し、その後システム開発とIT技術教育を主に行う。基幹業務、営業支援、Webデータベース、ECサイト、データマイニング（ビジネスインテリジェンス）をはじめとする大小数多くの開発プロジェクトに関わる。Java、C/C++、Oracle、MySQL、.NETなどによるシステム設計開発を長年行い、Linux系サーバ構築とメンテナンスも合わせて行う。近年 FlashによるRIAを発端として、Flex、AIRによる開発、著作、セミナー提供などを行う。技術者養成の専門スクールIT塾（Adobe 認定トレーニングセンター）校長としてFlex/AIRセミナーやIT技術講習も随時開講している。アドビCreative Suite Video Tutorialのローカリゼーション、Flex 4/AIR 2プレリリースプログラムコミュニティマネージャ、Flex/AIRハンズオンセミナーなど、アドビとのコラボレーションを多く行っている。

●Webシステム開発株式会社代表
http://www.websysd.com/

また、IT技術者のスキルアップを行うIT塾を運営。
Flash、AIR、Flex、Javaその他筆者が直接マンツーマン教育を行うこともある。

●IT塾
Adobe認定トレーニングセンター /CompTIAトレーニングセンター
http://www.it-school.org/

●取得資格
・Adobe認定 Flash/Flex/AIR インストラクタ
・Adobe Flexデベロッパー
・Adobe Flashデベロッパー
・Adobe Dreamweaverデベロッパー
・Sun認定Java ディベロッパ
・Sun認定Web コンポーネントディベロッパ
・Sun認定Solarisシステム管理者for Solaris 10
ほか

■本書サポートページ

http://isbn.sbcr.jp/59268/
本書をお読みいただいたご感想、ご意見を上記URLからお寄せください。

■注意事項

○本書内の内容の実行については、すべて自己責任のもとでおこなってください。内容の実行により発生したいかなる直接、間接的被害について、筆者およびソフトバンク クリエイティブ株式会社、製品メーカー、購入した書店、ショップはその責を負いません。

○本書の内容に関するお問い合わせなどは、ご面倒でも小社テクニカル書籍編集部まで、必ず書面にてご連絡くださいますようにお願いいたします。なお、質問の回答に関しては原則として著者に転送いたしますので、多少のお時間を頂戴、もしくは返答できない場合もありますのであらかじめご了承ください。
また、本書の内容以外についてのご質問については、お答えしかねますのでご了承ください。

フラッシュ アクションスクリプト ハンドブック

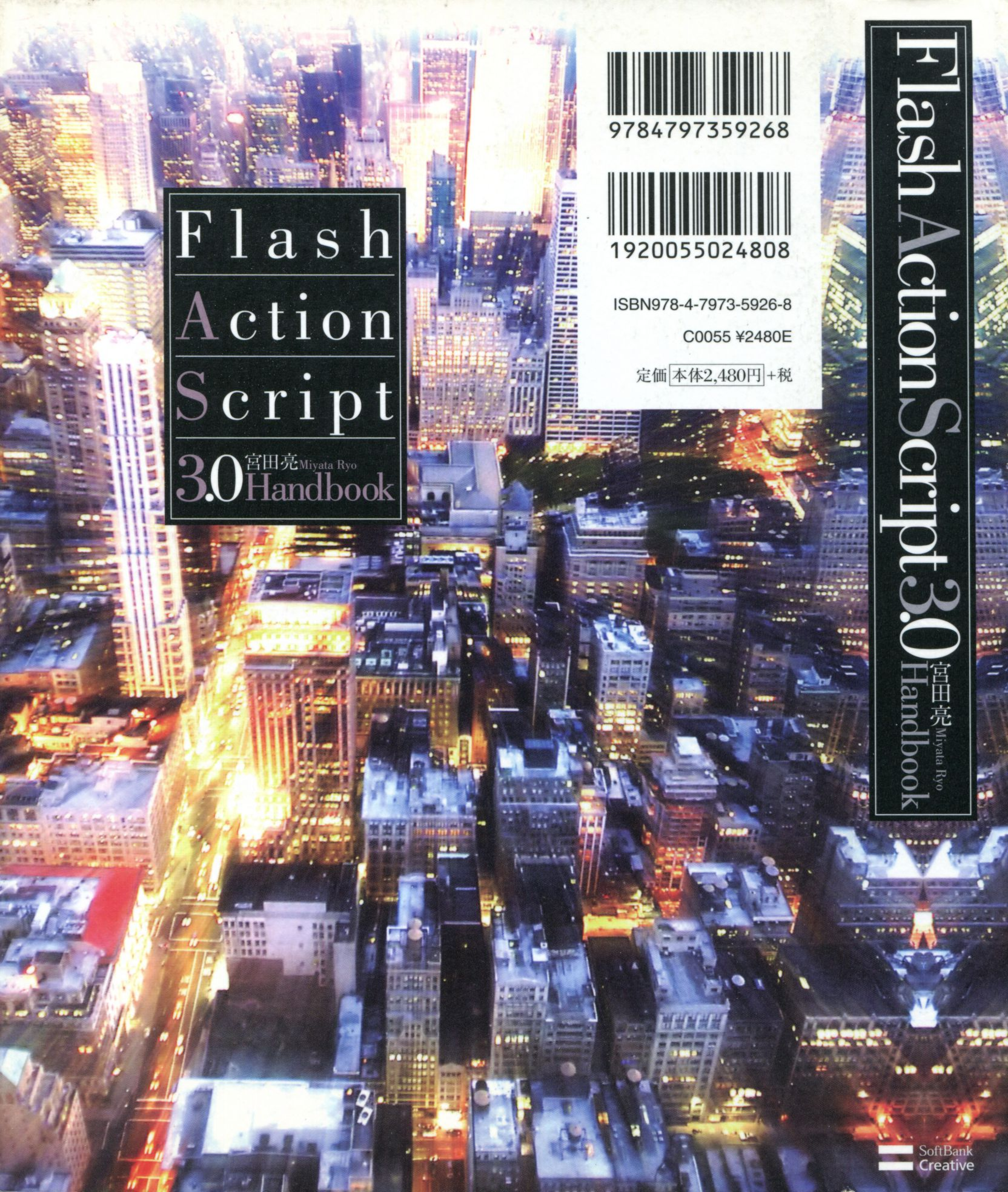
Flash ActionScript3.0 Handbook

2010年 7月 1日 初版発行

著　者…………宮田 亮（みやた りょう）
発行者…………新田 光敏
発行所…………ソフトバンククリエイティブ株式会社
〒107-0052 東京都港区赤坂4-13-13
03-5549-1201（販売）
カバーデザイン…………日下 充典
本文デザイン・組版……クニメディア株式会社
印　刷…………株式会社シナノ

落丁本、乱丁本は小社営業部にてお取り替えいたします。
定価はカバーに記載されております。

Printed in Japan ISBN978-4-7973-5926-8



Flash ActionScript 3.0 Handbook

宮田亮 Miyata Ryo

9784797359268
1920055024808
ISBN978-4-7973-5926-8
C0055 ¥2480E
定価 本体2,480円 + 税

Flash ActionScript 3.0 Handbook 宮田亮 Miyata Ryo

SoftBank Creative